Afstudeerverslag

Jort ten Donkelaar - CODE14

CODE14

fresh applications

CODE14

HBO-ICT Software Engineering, Saxion Hogeschool Enschede

Bedrijfsbegeleider: Herbert Altink

Afstudeerbegeleider: Peter Ebben

Auteur: Jort ten Donkelaar

Inhoudsopgave

1 Inleiding **5**

2 De opdracht **6**

2.1 Context 6

2.2 Probleemstelling 7

2.3 Opdrachtomschrijving 8

3 Onderzoek **10**

3.1 HubSpot 10

3.2 Privacy & anonimisering 12

3.3 API’s 13

3.4 Database & variabele datastructuur 15

3.5 Configuraties 20

3.6 Functioneel ontwerp 21

3.7 Requirements 24

4 Systeembeschrijving **28**

4.1 Architectuur 28

4.2 Front-end 29

4.3 Back-end 32

4.4 Database 37

4.5 Hosting 37

4.6 CI/CD 39

4.7 Testen 40

4.8 QA 40

5 Procesbeschrijving **42**

5.1 Plan van Aanpak 42

5.2 Onderzoek/ontwerp 42

5.3 Ontwikkeling 43

6 Conclusie **46**

6.1 Onderzoek 46

6.2 Implementatie 46

7 Aanbevelingen **48**

7.1 Configureerbare velden 48

8 Inhoudelijke reflectie **49**

9 Versiebeheer **50**

10 Literatuurlijst **51**

11 Bijlagen **52**

Bijlage 1: V2.0 Onderzoeksrapport (extern document) 52

Bijlage 2: User Stories (extern document) 52

Bijlage 3: Zelfreflectie (extern document) 52

1 Inleiding

Dit verslag bevat een beschrijving van de uitgevoerde afstudeeropdracht van Jort ten Donkelaar, voor de studie HBO-ICT Software Engineering aan het Saxion. De afstudeerstage is uitgevoerd bij CODE14 in Rijssen. De opdracht was het bouwen van een webapplicatie die een synchronisatie kon uitvoeren tussen twee accounts in het CRM systeem “HubSpot”, waarbij de nodige data wordt geanonimiseerd.

In dit verslag wordt eerst de opdracht verder uitgelegd op basis van een context en probleemstelling. Daarna wordt ingegaan op het onderzoek dat is uitgevoerd voor de stageopdracht. Dit onderzoek is voornamelijk gefocust op het uitzoeken van de mogelijkheden van de verschillende eisen en functionaliteiten waar de tool aan moet voldoen. Denk aan anonimisering, het aanroepen van HubSpot’s API’s en hoe slim kan worden omgegaan met meerdere synchronisaties naar 1 account.

Daarna wordt gefocust op het functioneel ontwerp en de requirements. Voor het functioneel ontwerp zijn onder andere wireframes, een database diagram en flowcharts gemaakt. De requirements bevat een lijst van eisen waar de tool aan moet voldoen, vanuit het opzicht van gebruikers (user), de toevoeging voor een bedrijf/organisatie (business) en vanuit technisch perspectief (systeem).

Vervolgens wordt meer gefocust op de implementatie van de synchronisatietool. Er wordt uitgelegd welke frameworks zijn gebruikt en waarom, hoe de architectuur er uit ziet, hoe de UI er uiteindelijk uitziet en hoe de backend werkt in de zin van het verwerken van API’s en het uitvoeren van de synchronisatie zelf. Daarbij wordt ook gekeken naar tests en QA (Quality Assurance).

Verder is er een procesbeschrijving, die uitlegt wat de reden was achter verschillende processen die zijn uitgevoerd tijdens de stage, en daarbij wat er precies is uitgevoerd en hoe. Daarna is er een conclusie die de antwoord geeft op vragen van het onderzoek en die vertelt wat er precies is afgekomen bij de bouw van de synchronisatietool. Ten slotte is er nog een aanbeveling gedaan voor een eventuele andere mogelijkheid van het implementeren van deze tool.

2 De opdracht

2.1 Context

CODE14 is een softwarebedrijf dat werkt aan verschillende (maatwerk) ICT-oplossingen die bedrijven verder kunnen helpen. Dit zijn oplossingen als web-apps, mobiele apps, websites, API-koppelingen, co-creatie software etc. Zij focussen zich niet op een bepaalde specifieke branche, maar willen met elk type bedrijf om de tafel om te kijken of ze hun verder kunnen helpen en om zelf ook nieuwe kennis op te doen over processen en markten.

Het verder helpen van een organisatie in hun proces gaat vaak niet alleen puur om het bouwen van een software-product. CODE14 raadt bijvoorbeeld hun klanten soms ook aan om bepaalde bestaande (externe) software tools/producten te gaan gebruiken, waar CODE14 dan vaak ook koppelingen aan kan bieden. Een van die producten is HubSpot. HubSpot is een CRM (Customer Relationship Management) systeem uit Amerika waar een organisatie onder andere hun contacten, verkopen, producten en marketing kan bijhouden. Daarbij bevat het ook een CMS (Content Management Systeem) om websites te bouwen of te vullen met data uit HubSpot.

Het komt voor dat klanten graag hun eigen systemen willen koppelen aan HubSpot. Dit omdat ze bijvoorbeeld geregistreerde contacten in een eigen applicatie graag ook in HubSpot willen bijhouden. Het mooie voor CODE14 aan HubSpot is dat het een uitgebreid developers platform biedt waarbij veel functionaliteiten uit HubSpot via API's aangeroepen kunnen worden. Dit maakt zulke koppelingen met eigen systemen in veel situaties mogelijk. Zo heeft CODE14 in het verleden vaker gebruik gemaakt van de API's in HubSpot om zulke koppelingen te bouwen.

Bij het bouwen van zo'n koppeling wordt gebruik gemaakt van OTAP omgevingen (ontwikkel, test, acceptatie & productie). Zo'n scheiding zorgt ervoor dat een klant nooit wat zou merken van tests die een ontwikkelaar uitvoert. In een ontwikkelomgeving staan de laatste veranderingen van een ontwikkelaar, en deze wordt regelmatig geüpdatet. Als het project zo ver is ontwikkeld dat een relatief stabiele versie staat, kan deze gedeployed worden in een testomgeving waar het project door andere ontwikkelaars kan worden getest. Ondertussen kan een ontwikkelaar gewoon door ontwikkelen in zijn ontwikkelomgeving. Als de test versie in de testomgeving is goedgekeurd, kan deze worden goedgekeurd door de klant in een acceptatie-omgeving. Deze staat dus als brug tussen test en productie, en wordt vaak met wat duidelijkere data gevuld. Als de klant de acceptatieomgeving versie goedkeurt, kan het project naar productie.

Deze scheiding tussen O/T/A & P komt in de context van HubSpot ook voor. Het daadwerkelijke HubSpot account van een klant (account = omgeving) kan worden gezien als de productieomgeving. Om de koppeling te kunnen testen in een ontwikkelomgeving wordt altijd een apart HubSpot account aangemaakt, speciaal voor het testen van de koppeling in de ontwikkelomgeving. Hetzelfde gebeurd voor de test-omgeving en de acceptatie-omgeving, omdat het niet handig is om alle omgevingen van koppelingen aan eenzelfde account te koppelen. Het wordt dan namelijk een stuk onduidelijker om te zien welke data waar vandaan komt.

2.2 Probleemstelling

Bij HubSpot bestaan er 3 verschillende soorten accounts: een regulier account, een developers account en een test account. Een regulier account is een normale omgeving binnen HubSpot, zoals een bedrijf deze zou aanmaken. Binnen zo'n account kunnen verschillende pakketten worden aangeschaft om meer functionaliteiten te verkrijgen, en deze pakketten hebben vaak ook verschillende niveaus (starter, professional etc.). Vervolgens is er een developers account. Dit is de basis voor developers-platform van HubSpot. Het is geen omgeving binnen HubSpot, maar wordt gebruikt om apps te ontwikkelen voor HubSpot. Het staat dan ook los van de reguliere HubSpot accounts; er kan een developers account worden aangemaakt zonder dat een gebruiker een regulier account heeft, en andersom ook. Onder een developers account kunnen onder andere test accounts worden gemaakt. Een testaccount is een HubSpot omgeving net als een regulier account, maar bevat gratis alle hoogste pakketten die aan te schaffen zijn binnen HubSpot. Deze accounts zijn bedoeld voor ontwikkelaars om de koppelingen met de HubSpot API's te kunnen testen. Normaal gesproken maakt de klant zijn reguliere account aan, en maakt CODE14 een developer account. Er kunnen wel meerdere gebruikers worden gekoppeld aan een enkel account, dus iemand bij CODE14 zou wel toegang kunnen krijgen tot de HubSpot omgeving van een klant.

Het probleem is hier echter dat deze testaccounts standaard helemaal leeg zijn. Er zijn binnen HubSpot geen mogelijkheden om bijvoorbeeld een bestaande omgeving te kopiëren/dupliceren naar een testaccount, of om in een keer veel data te exporteren en importeren naar een nieuwe omgeving. Dit is vaak wel wenselijk voor ontwikkelaars, omdat er anders niet goed geverifieerd kan worden of bepaalde koppelingen met HubSpot goed werken. Daarbij zou het voor de klanten ook wenselijk zijn omdat ze anders minder goed kunnen zien dat bepaalde functionaliteiten goed werken.

Om een testaccount te kunnen vullen met data zal er dus een tool moeten worden gebouwd die middels de HubSpot API's data ophaalt uit de productieomgeving van de klant en deze invult in een ontwikkelomgeving. Daarbij spelen nog een aantal dingen een rol. Allereerst is er een probleem van privacy. Gezien de productieomgeving van de klant privacygevoelige data bevat, zoals persoonsgegevens, is het niet de bedoeling dat deze letterlijk in een testomgeving worden geplaatst. Om deze data alsnog te gebruiken zal deze dus geanonimiseerd moeten worden. De reden waarom er voor anonimisering is gekozen tegenover het volledige genereren van data is omdat allereerst niet alles geanonimiseerd hoeft te worden, maar ook vooral omdat je dan werkt met situaties die te verwachten zijn. Elke klantomgeving ziet er namelijk anders uit en is ook op een andere manier ingevuld. Het kan bijvoorbeeld zijn dat bepaalde velden in sommige gevallen leeg worden gelaten door bepaalde klanten, of op een specifieke manier worden ingevuld. De anonimisering is er dus om die situatie in opbouw, structuur en hoeveelheid van die datavelden zoveel mogelijk te behouden. Welke velden precies op welke manier geanonimiseerd moeten worden zal in eerste instantie van tevoren moeten worden bepaald door te kijken naar de properties binnen HubSpot. In een ideale situatie zou dit echter door een ontwikkelaar geconfigureerd moeten kunnen worden, door aan te geven welke velden een klant gebruikt en wat voor soort data daarvoor gegenereerd moet worden.

Een ander probleem is de uitgebreide en complexe structuur van HubSpot. Veel van de meest gebruikte CRM objecten, zoals contacten en bedrijven, zijn vrij flexibele objecten. Het is namelijk aan de gebruiker om te bepalen welke data hij precies bij deze objecten wil bijhouden. Naast bijvoorbeeld wat standaardgegevens als een naam, mailadres en telefoonnummer kunnen allerlei zogenoemde “properties” worden toegevoegd. Ook is het mogelijk voor een gebruiker om zelf zo’n dataveld aan te maken, mocht er een veld niet beschikbaar zijn. Het lastige daarbij is dus dat je als ontwikkelaar van tevoren niet weet welke objecten je gaat ontvangen bij een request. Ook kan HubSpot deze zelfgemaakte properties niet verwerken bij het aanmaken van een object als deze properties niet eerst worden aangemaakt. Er moet dus bij het invullen van de data rekening gehouden worden dat eerst de properties van het object worden aangemaakt.

Andere problemen die voor kunnen komen zijn bijvoorbeeld limieten m.b.t. API’s. Er zitten namelijk limieten bijvoorbeeld op het aantal API calls per dag, per 10 seconden en het aantal objecten dat je in een keer mag opvragen/invullen. Daarbij moet er worden nagedacht hoe OTAP moet worden ingericht. Zoals in de context is beschreven worden er door de ontwikkelaars bij CODE14 voor een klant test-accounts aangemaakt binnen HubSpot in de vorm van een ontwikkel, test en acceptatie account. De synchronisatietool (de tool die gemaakt is als afstudeeropdracht om dit synchronisatie probleem op te lossen) moet kunnen zien dat het hoofdaccount van de klant het productie account is, en dat de overige accounts development omgeving zijn.

2.3 Opdrachtomschrijving

De opdracht is om een tool te bouwen die de productiedata uit een HubSpot omgeving kan ophalen, deze data kan anonimiseren en vervolgens de geanonimiseerde data kan invullen in een testomgeving. De tool moet op een centrale server worden gedraaid die beschikbaar is voor alle ontwikkelaars bij CODE14. Binnen de synchronisatietool moeten klanten kunnen worden toegevoegd, met daaronder de verschillende HubSpot-accounts die worden gebruikt voor een klant (de productieomgeving met daarbij de testomgevingen). Deze lijst van klanten en omgevingen moeten eenvoudig, zonder code wijzigingen, configureerbaar zijn zodat de synchronisatie zo generiek mogelijk is. Generiek in de zin dat de tool voor elke klant en elke omgeving inzetbaar is, en dat er in zoverre rekening wordt gehouden met de structuur en de inrichting van de data van een klant zodat deze op eenzelfde manier wordt geanonimiseerd. De data die nodig is bij de configuratie van klanten zijn echter wel gevoelige gegevens zoals sleutels/secrets. Er zal dus nagedacht moeten worden over het encrypten van deze data.

Bij het starten van een synchronisatie moet gezorgd worden dan de tool geauthenticeerd API calls kan doen. Dit kan worden gedaan door middel van een API key, of een OAuth access token als de HubSpot omgeving daarmee beveiligd is. Vervolgens moet de tool data op kunnen halen uit de productieomgeving, vergelijken met opgeslagen data uit een eerdere synchronisatie naar de doelomgeving om de gewijzigde objecten te herkennen, en vervolgens invullen in de gegeven testomgeving. Dit proces moet waar mogelijk asynchroon kunnen draaien, maar er moet goed rekening gehouden worden met een volgorde van synchronisatie. Het kan namelijk zijn dat bepaalde objecten voor andere aangemaakt moeten worden in verband met bijvoorbeeld koppelingen. Gezien er op momenten veel API calls tegelijkertijd zullen gebeuren moet er ook rekening gehouden worden met rate limits.

Als je namelijk constant API calls blijft doorsturen terwijl HubSpot niks meer kan verwerken dan mislukt uiteraard de synchronisatie. Daarbij kan een eventuele externe koppeling met hetzelfde account ook geen API calls meer doen, wat dus verschillende processen kan verstoren. Een externe koppeling kan bijvoorbeeld een andere applicatie zijn die automatisch nieuwe contacten in HubSpot aanmaakt vanuit een extern systeem.

Het kan voorkomen dat een ontwikkelaar meerdere keren een synchronisatie start tussen twee accounts. In dat geval wil er dus voor zorgen dat je alleen de velden ophaalt en verwerkt die echt gewijzigd zijn. Daarvoor is het handig om een database te gebruiken waar de gesynchroniseerde data in opgeslagen wordt.

Aangezien het bij de synchronisatie in veel gevallen gaat om privacygevoelige informatie moet er dus gezorgd worden dat deze wordt geanonimiseerd. Daarbij moet het wel duidelijk blijven om welk object het vanuit de productieomgeving gaat, omdat je er anders niet achter kan komen welke data nieuw/gewijzigd is. Hier moet dus rekening mee gehouden worden bij het ontwerpen en inrichten van de database.

Een ander ding waar de database en de tool mee om moet kunnen gaan zijn de variabele datavelden die beschikbaar zijn in HubSpot. Deze zijn cruciaal voor het invullen van de inhoudelijke data, gezien de daadwerkelijke velden moeten bestaan om ze te kunnen vullen. Er moet dus gezorgd worden dat deze vooraan het synchronisatieproces worden gesynchroniseerd, en dat er een plek komt binnen de database om deze velden op te slaan.

3 Onderzoek

3.1 HubSpot

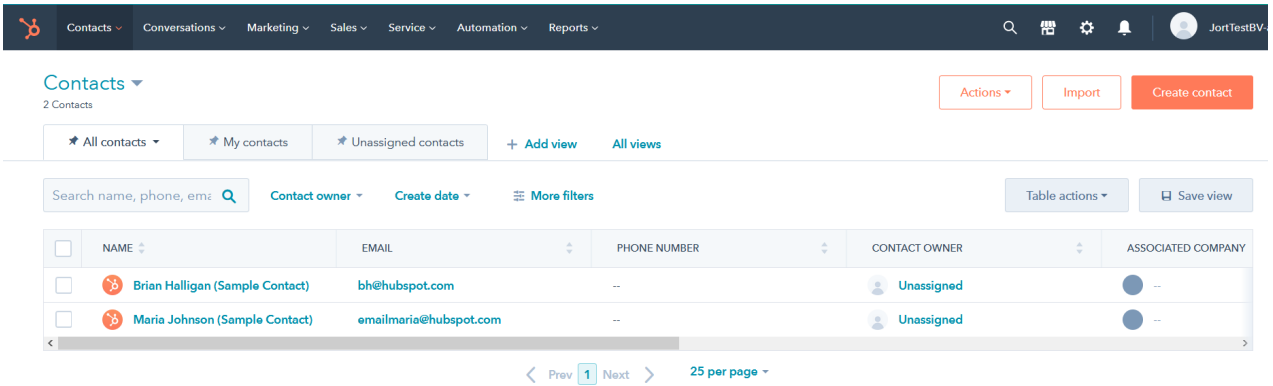
3.1.1 Aanleiding

Voor het bouwen van de synchronisatietool was het belangrijk om te weten wat HubSpot precies is, en hoe het in elkaar zit. Daarbij was het nodig om een overzicht te hebben van welke pagina's/data binnen HubSpot veel gebruikt worden door klanten, en daarbij voornamelijk de koppelingen tussen HubSpot en de systemen van de klant. Zo werd het namelijk duidelijk welke pagina's de synchronisatietool in elk geval moest gaan synchroniseren. De onderzoeksvraag hierbij is: "Welke invloed hebben de verschillende pakketten en accounts van HubSpot omgevingen?". Deze vraag is zo opgesteld omdat het hiermee duidelijk zou worden hoe accounts binnen HubSpot werken, welke configuraties een account achter de schermen heeft (OAuth, verschillende pakketten etc.) en hoe developers/testaccounts werken in HubSpot. Daarbij was het voor mij van tevoren nog niet duidelijk hoe HubSpot precies werkte, dus het aanmaken van een eigen HubSpot account was een goed begin om HubSpot te leren kennen, en dit leidde uiteindelijk tot het onderzoeken van de verschillende pagina's binnen HubSpot.

HubSpot is een CRM platform voor marketing, sales en services gemaakt door het gelijknamige Amerikaanse softwarebedrijf. Het bedrijf is gericht op het helpen van bedrijven om verschillende processen binnen een bedrijf soepeler te laten lopen, zoals marketing en verkopen. Het is een webapplicatie, en wordt gehost door HubSpot zelf.

3.1.2 Resultaat

HubSpot bestaat uit 4 verschillende pakketten: Sales, Marketing, Service en CMS. Het sales pakket is gericht op verkoop-functionaliteiten. Hier kan een bedrijf onder andere zijn producten in registreren, en een pipeline bijhouden van deals. Het marketingpakket bestaat uit verschillende marketing opties, zoals e-mails, social media en advertenties. Het servicepakket bestaat uit functionaliteiten voor interactie met klanten, bijvoorbeeld via email of via een live-chat. Ook kunnen daar tickets/issues worden bijgehouden die zijn voorgekomen. Ten slotte is er een CMS systeem, waarmee websites kunnen worden opgebouwd en worden ingevuld. Hieronder is een voorbeeld te zien van een pagina binnen HubSpot, om precies te zijn de "Contacts" pagina, met twee aangemaakte contacten.



Figuur 1: Contacts pagina in HubSpot

Behalve voor het CMS pakket zijn er gratis pakketten beschikbaar. Daarnaast heeft een bedrijf drie opties om elk pakket los te upgraden naar een Starter, Professional of Enterprise niveau. Elke stap hoger biedt verschillende extra mogelijkheden, en vaak ook een ophoging van de limiet aan records dat kan worden opgeslagen.

Ten slotte biedt HubSpot mogelijkheden voor developers via hun developers-platform. Veel van de functionaliteiten in HubSpot zijn opengesteld via een API, waardoor het dus mogelijk is om HubSpot te koppelen aan bijvoorbeeld een eigen stuk software van een klant. Hierdoor kan een klant bijvoorbeeld automatisch zijn contacten en verkopen synchroniseren. Binnen het developers-platform kan een ontwikkelaar dan test-accounts aanmaken, wat eigenlijk HubSpot omgevingen zijn met toegang tot alle hoogste pakketten. Hiermee kan een ontwikkelaar dus vrij zijn API koppelingen testen, zonder belemmerd te worden of zelf te belemmeren. Dit is ook waar CODE14 gebruik van maakt bij het bouwen van koppelingen tussen HubSpot en systemen van de klant.

Het zou uiteindelijk niet uit moeten maken welke pakketten een klant heeft. Voor de ontwikkeling zijn alle maximale pakketten al beschikbaar, dus daar zouden nergens belemmeringen moeten zitten bij het aanroepen van de API's. Sterker nog, omdat test-accounts de maximale pakketten hebben zijn de rate limits ook het hoogst ingesteld, wat goed is omdat je vaak meerdere API calls maakt naar de doelomgeving. Stel dat een klant een gratis HubSpot pakket heeft, dan zou dit niet uit moeten maken voor de synchronisatietool omdat de API's op dezelfde manier blijven werken. Klanten zelf kunnen dan bijvoorbeeld minder objecten aanmaken, maar er missen geen belangrijke functionaliteiten om de synchronisatie niet uit te kunnen voeren.

3.2 Privacy & anonimisering

3.2.1 Aanleiding

Het onderzoeken naar privacy en anonimisering is opgesplitst in twee deelvragen:

1. “Welke informatie binnen HubSpot is voor een klant privacygevoelig?”
2. “Hoe kan data van een klant worden geanonimiseerd waarbij het duidelijk blijft om welke soort data het gaat?”.

De eerste deelvraag gaat voornamelijk over de data die binnen HubSpot wordt bijgehouden, en wat daarvan voor klanten privacygevoelig is. Hiervoor is niet naar alle individuele klanten gekeken, maar is een globale analyse gedaan die voor elke klant geldt. Dit heb ik voornamelijk gedaan omdat in de basisversie van de synchronisatietool de anonimisering generiek wordt gedaan voor elke synchronisatie van elke klant. Het doel is wel om uiteindelijk de anonimisering configurabel te maken vanuit de synchronisatietool, maar in de basis moet de synchronisatietool dus de belangrijkste velden al wel anonimiseren. Het doel van deze deelvraag is dus om aan te geven welke velden het belangrijkste zijn. Het is daarbij dus de bedoeling dat uiteindelijk de configuratie van die anonimisering vanuit de synchronisatietool wordt gedaan, en dat hier niet voor elke klant een codewijziging voor moet gebeuren.

De tweede deelvraag gaat over hoe de anonimisering precies moet gaan gebeuren. Hiervoor is gekeken naar een aantal verschillende anonimiseringstechnieken, waarbij uiteindelijk vrij snel duidelijk werd dat het gebruiken van een data-faker genoeg zou zijn.

3.2.2 Resultaat

Binnen HubSpot gaan allerlei verschillende soorten data rond: contactpersonen, bedrijfsinformatie, verkoopdata etc. Elk van deze dataobjecten bestaat dan weer uit een groot aantal velden. Een contact bestaat bijvoorbeeld uit een voornaam/achternaam, adresgegevens, telefoonnummer etc. Volgens de AVG mogen bijvoorbeeld persoonsgegevens niet (ongevraagd) verwerkt worden. Daarnaast is het vanuit klanten specifiek niet wenselijk om bijvoorbeeld alle deals en producten letterlijk over te nemen. Het is daarom de bedoeling dat deze data waar nodig geanonimiseerd wordt, zodat de data alsnog wel gebruikt kan worden. In het onderzoeksrapport (zie Bijlage 1) staat een analyse die per object aangeeft wat het belangrijkste is om te anonimiseren en wat het minst. Dit is bepaald op wat klanten zelf het meest gebruiken en wat door ontwikkelaars het meest wordt gebruikt in het bouwen van koppelingen met andere systemen.

Er bestaat de mogelijkheid om data niet te anonimiseren en om gewoon rechtstreeks datasets te genereren. Het probleem is echter dat je dan werkt met een dataset die enigszins te verwachten is. Je genereert dan namelijk data altijd op dezelfde manier, terwijl het bijvoorbeeld voor kan komen dat een klant bepaalde velden niet altijd invult, of dat ze een specifiek (zelfgemaakt) dataveld altijd gebruiken. Het is vanuit het perspectief van de ontwikkelaars dan wenselijk dat ze hun API koppelingen kunnen afstemmen op de structuur die de klant gebruikt binnen HubSpot, en dat gaat lastiger als ze altijd een soortgelijke dataset hebben. Uiteraard is het uiteindelijk alsnog mogelijk om voor elke klant een

specifieke data faker op te zetten op basis van de structuur van hun data, maar dat is eigenlijk wat er nu ongeveer wordt gedaan door CODE14. Properties worden handmatig aangemaakt op basis van de structuur van de klant-omgeving, en data wordt ook handmatig ingevuld. Dit kost veel tijd, en is alsnog ingewikkeld om op te zetten omdat er veel verschillende situaties zijn die nagebootst moeten worden (lege velden, hoeveelheid van objecten, variabele velden moeten precies worden nagemaakt etc.). Daarbij kunnen objecten binnen HubSpot aan elkaar gekoppeld zitten: een contact kan bijvoorbeeld gelinkt zijn aan een bedrijf, of een ticket aan een deal. Ook daar is het handiger om die structuur over te nemen van de klant in plaats van deze zelf te verzinnen, omdat dat anders vreemde resultaten kan weergeven. Het kan daarbij ook zijn dat een klant regelmatig zijn properties aanpast. In dat geval wil je niet elke keer aan een klant vragen of er iets gewijzigd is, maar kan je beter een synchronisatie draaien om de laatste wijzigingen automatisch op te halen. Ten slotte wil je ook dat het aantal records dat een klant heeft ongeveer overeenkomt, zodat de ontwikkelaars goed kan anticiperen op het aantal objecten dat zij kunnen verwachten.

Voor het anonimiseren van data is onderzoek gedaan naar verschillende anonimiseringstechnieken. Het is namelijk belangrijk dat de data nog wel leesbaar is en bijvoorbeeld niet wordt versleuteld waarbij het niet meer duidelijk is of het gaat om een naam, emailadres etc. Uiteraard zou dit voor de software niet uitmaken, maar een ontwikkelaar wil die data weer gebruiken om zijn koppeling mee te testen, dus dan wil diegene wel kunnen zien of de data logisch is overgenomen. Daarbij is het sowieso ongewenst om de geanonimiseerde data terug te kunnen draaien naar de productiedata. Het bijhouden van een ID van de productieomgeving is genoeg om de synchronisatie te kunnen voltooien. Daarom is gekozen om elk veld dat geanonimiseerd moet worden opnieuw te genereren met een data-faker. Hiervan zijn er twee bekeken. De eerste is de PHP library “Faker” gemaakt door een gebruiker genaamd “fzaninotto” op GitHub. Deze is gebaseerd op oudere Fakers, bijvoorbeeld voor de talen Perl en Ruby, en wordt op het moment van schrijven nog het meest gebruikt door PHP ontwikkelaars. Ook bij CODE14 wordt deze faker regelmatig ingezet, en dit is ook waarom deze als eerste is bekeken. Functioneel was deze erg goed, maar de ontwikkelaar heeft rond december 2020 besloten om de library niet meer te gaan ondersteunen. Dit kan er op den duur voor zorgen dat bij nieuwe versies van PHP deze library niet meer gaat werken. Er is daarom gekozen voor de volgende beste optie, namelijk FakerPHP. Deze is op dezelfde manier opgezet als de Faker van fzaninotto en is ook gebaseerd op de Perl en Ruby Fakers. Deze wordt nog steeds ondersteund en toont nog geen plannen om ondersteuning te stoppen. Er is door mij nog wel verder gezocht naar andere opties, maar FakerPHP voldoet aan de eisen om in elk geval de standaard velden uit HubSpot te kunnen anonimiseren en daarom is daarvoor gekozen.

3.3 API’s

3.3.1 Aanleiding

Dit onderdeel van het onderzoek is gericht op het onderzoeken van de HubSpot API’s: hoe deze in elkaar zitten, wat de limitaties zijn etc. De onderzoeksvraag die hierbij hoort is: “Welke HubSpot API’s kunnen het beste gebruikt worden voor de synchronisatie van data?”. Deze geeft het doel duidelijk weer, namelijk het in kaart brengen van de verschillende API’s die nodig zullen zijn bij het uitvoeren van de synchronisatie. Hierdoor kan bijvoorbeeld ook

gekeken worden wat er niet kan met de API's, omdat niet alle functionaliteiten beschikbaar zijn met een API. Ook moet hier gekeken worden hoe OAuth wordt ingezet bij het aanroepen van de API's, en hoe deze tokens verkrijgbaar zijn. Dit onderzoek is voornamelijk uitgevoerd door de API documentatie van HubSpot (HubSpot APIs | Usage guidelines, z.d.) door te lezen, en door te experimenteren met de verschillende API's middels Postman. Postman is een tool speciaal ontwikkeld voor creëren, testen en documenteren van (restful) API's.

3.3.2 Resultaten

API's binnen HubSpot zijn gebaseerd op REST conventies. Ze gebruiken veel standaard HTTP features zoals de standaard methoden (POST, GET etc.) en error response codes. HubSpot kent verschillende versies bij haar API's, waar in de meeste gevallen V3 de nieuwste en daarom ook de standaard is. De API's sturen altijd een response in JSON formaat, en zijn ontworpen om een voorspelbare URL structuur te geven. Dit is voornamelijk te zien bij de verschillende CRM object API's: de structuur voor de API's (v3) is bij contacts, companies, tickets, deals, line items en products allemaal vrijwel hetzelfde. De base-URL voor API's is https://api.hubapi.com. Daarna volgt vaak de categorie waar de API onder valt (CMS = /content, CRM = /crm, automation = /automation etc.). Daarna komt de versie van de API (/v1, /v2 of /v3), en dan of een subcategorie of de specifieke API naam (bij CRM objecten: /objects/:objectType, bij CRM pipelines: direct /pipelines).

Het aangeven van welk account geraadpleegd moet worden om de API query op uit te voeren kan op twee manieren. Standaard is een account voor API's alleen afgeschermd met een API key. Deze is te vinden in de configuraties van een account, en verloopt nooit. Deze kan dan worden meegestuurd als parameter van de API en daarmee weet HubSpot waar hij zijn data moet ophalen. Daarnaast kan een account verder worden beveiligd d.m.v. OAuth. Als dit specifiek wordt aangezet kan je geen requests meer sturen met een API key. Als je de API's wilt oproepen moet een gebruiker handmatig naar een specifieke URL gaan. Daar moet toestemming worden gegeven voor het koppelen van het HubSpot account en de bron waar de API call vanuit gedaan moet worden. De bron van de API krijgt dan een autorisatie-code, waarmee een Access Token en een Refresh Token mee op te halen zijn. Als deze Access Token dan wordt meegestuurd van de API, en deze is niet verlopen, is de API geautoriseerd en weet HubSpot ook gelijk bij welk account die token hoort.

Het aanroepen van de API's is echter niet ongelimiteerd. Elk HubSpot account heeft op de achtergrond een teller die bijhoudt hoeveel API calls er op een dag zijn gedaan. Verschillend per pakket zit er een dagelijks limiet op het aanroepen van die API's, en zodra die overschreden is kunnen er dus geen calls meer gedaan worden. Een gebruiker zelf kan dan echter nog wel handmatig acties uitvoeren binnen zijn account. Naast de dagelijkse limiet is er ook een "burst" limiet, waar je op basis van welk pakket je hebt maar een aantal API calls per 10 seconden mag doen. Dit 10-seconden limiet verschilt ook nog per API: sommige zwaardere APIs mogen bijvoorbeeld minder vaak worden aangeroepen in korte tijd dan andere. Omdat de synchronisatietool meestal niet de enige applicatie is die API calls uitvoert, moet er dus gezorgd dat er binnen die limieten wordt gebleven, door bijvoorbeeld maar de helft van de limiet te hanteren. De limieten worden hieronder beschreven in figuur 2.

Product tier	Limits
Free and Starter	Burst: 100/10 seconds Daily: 250,000
Professional and Enterprise	Burst: 150/10 seconds Daily: 500,000
API Add-on (any tier)	Burst: 200/10 seconds Daily: 1,000,000

Figuur 2: API Rate limits voor HubSpot per pakket-niveau

3.4 Database & variabele datastructuur

3.4.1 Aanleiding

Het onderzoek naar het gebruik van een database en het verwerken van variabele datastructuur bestaat uit twee verschillende deelvragen:

1. Hoe kan er worden omgegaan met variabele datastructuur uit HubSpot?
2. Hoe kan met behulp van een database-tussenlaag bepaald worden welke data gewijzigd moet worden in de te synchroniseren omgeving?

De eerste deelvraag gaat over het rekening houden met de variabele datastructuur binnen HubSpot tijdens de synchronisatie. Dit gaat over een aantal verschillende componenten, met name de properties die onder de verschillende CRM-objecten zitten en de pipelines & pipeline stages. Het was nodig om hier onderzoek naar te doen omdat het duidelijk moest worden of- en hoe deze variabele datastructuur opgehaald kan worden, en hoe hier in de rest van de synchronisatie rekening mee gehouden moest worden. Zonder het verwerken van deze variabele datastructuur werkt de synchronisatie namelijk maar beperkt, en kan het maar een aantal velden per object synchroniseren.

De tweede deelvraag gaat over het inzetten van de database in het synchronisatieproces. Het was nodig om hier onderzoek naar te doen om eens goed na te denken over hoe de database er precies uit moest gaan zien. Er zijn namelijk een aantal belangrijke velden die moeten worden opgeslagen zodat objecten herkent kunnen blijven worden, maar ook een aantal velden die encrypted opgeslagen moeten worden. Het gebruik van de database helpt namelijk om een tweede synchronisatie naar een specifiek account efficiënter te maken, omdat je dan alleen de wijzigingen op hoeft te halen. Daardoor moeten er echter wel een aantal specifieke waarden worden opgeslagen, zoals ID’s uit de productieomgeving, datums waarop ze gewijzigd zijn etc.

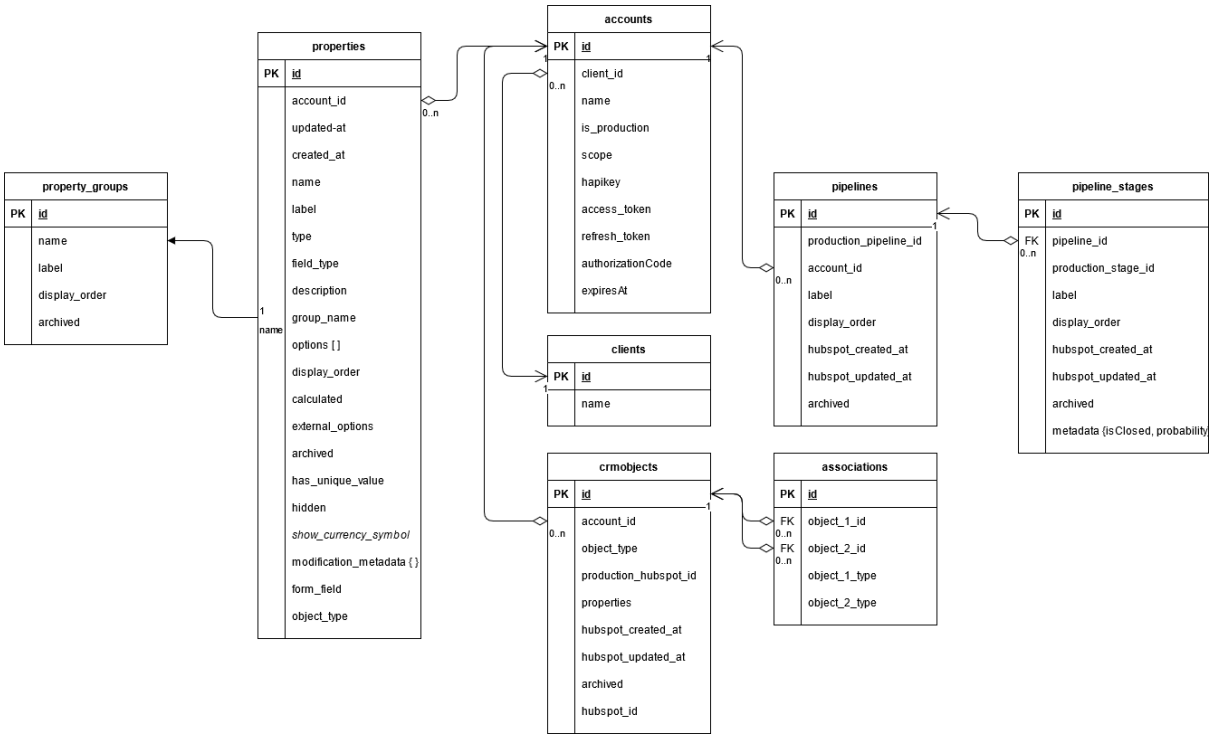
3.4.2 Resultaat

3.4.2.1 Database

Voor het uitvoeren van synchronisaties is het handig om een database in te zetten. Hier zijn meerdere redenen voor. De eerste keer dat de data van een account wordt gesynchroniseerd naar een ander account is er nog niks aan de hand. Het doel-account is vaak nog leeg, en de synchronisatietool hoeft dus weinig rekening te houden met wat er in die omgeving staat. Het kan echter zijn dat je later het testaccount wilt updaten met de laatste data uit het productie-account. In dat geval wil je dus alleen weten wat de wijzigingen zijn met de omgeving waar je naartoe wilt synchroniseren. Anders ga je overbodig veel data ophalen uit de productieomgeving, en stop je allemaal dubbele data in de testomgeving. In dat geval is het het handigst om een database op te zetten waar je tijdens de synchronisatie je data in kwijt kan. Bij een nieuwe synchronisatie hoef je alleen maar de laatst gewijzigde/toegevoegde data op te halen uit de productieomgeving, en hoef je ook alleen maar die data door te voeren naar je testomgeving. Hiervoor heeft HubSpot een speciale API ontwikkeld voor alle CRM objecten, namelijk de “Search/Filter/Group” API’s. In deze search API’s kun je lijsten data ophalen op dezelfde manier als de normale GET API, maar hier kunnen dus o.a. filters aan toegevoegd worden. Een filter wordt in dit geval dan dat de “last modified date” groter moet zijn dan de nieuwste wijzigingsdatum uit de database. Dit scheelt ontzettend veel API calls, en ook veel tijd.

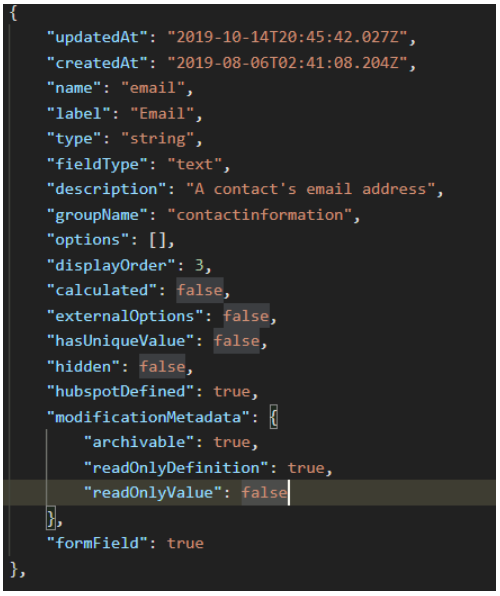
Een andere reden is dat je vaak meerdere testaccounts op zet voor het bouwen van een koppeling voor een klant, vaak in elk geval 3 (ontwikkel, test, acceptatie). Als je al voor een van die omgevingen data hebt gesynchroniseerd zou je in plaats van nog een keer alle data uit HubSpot op te halen eerst de data uit de database kunnen kopiëren.

Hieronder in figuur 3 is het database schema/diagram te zien:



Figuur 3: Database diagram synchronisatietool

De database bestaat (voor de synchronisatie) uit 8 tabellen: property_groups, properties, accounts, clients, crmobjects, pipelines, associations en pipeline_stages. “properties” en “property_groups” bevatten alle informatie over alle datavelden die onder een CRM object uit HubSpot bestaan, en de groep waar ze bijhoren. Een property is bijvoorbeeld het veld “email” onder een contact. Deze hoort onder de propertygroup “contactinformation”, en bevat verder een hele hoop extra data die bijvoorbeeld het type data aangeven. De CRM objecten (contacten, bedrijven etc.), de properties en de pipelines zijn in dit geval ook de enige dataobjecten/pagina’s die gesynchroniseerd zullen worden. Andere HubSpot functionaliteiten zoals mails, websites e.d. worden in eerste instantie nog niet gedaan, maar zouden wel eenvoudig kunnen worden toegevoegd aan de database door een nieuwe tabel aan te maken en deze ook te koppelen aan de account-tabel.



Figuur 5: Postman, response object van properties API HubSpot

“clients” bevat alle klanten die geregistreerd zijn vanuit de UI van de synchronisatietool. In de database zullen dus gesynchroniseerde omgevingen staan van meerdere verschillende klanten. Daarbij zijn “accounts” de HubSpot accounts die onder de klant zijn gekoppeld. Deze bevatten o.a. (encrypted) OAuth tokens of een API key voor HubSpot.

“crmobjects” is een verzameltabel voor alle CRM objecten die gesynchroniseerd moeten worden, dus de contacten, bedrijven, deals etc. De mogelijkheid bestond om deze in losse tabellen op te splitsen, maar HubSpot heeft deze objecten zo opgezet dat ze er in de basis precies hetzelfde uit zien. Qua inhoud verschillen ze alleen bij de properties en deze zijn vaak toch variabel, dus deze worden in zijn geheel als JSON formaat opgeslagen in de tabel. Zo is er voorkomen dat deze tabellen te veel op elkaar zouden lijken.

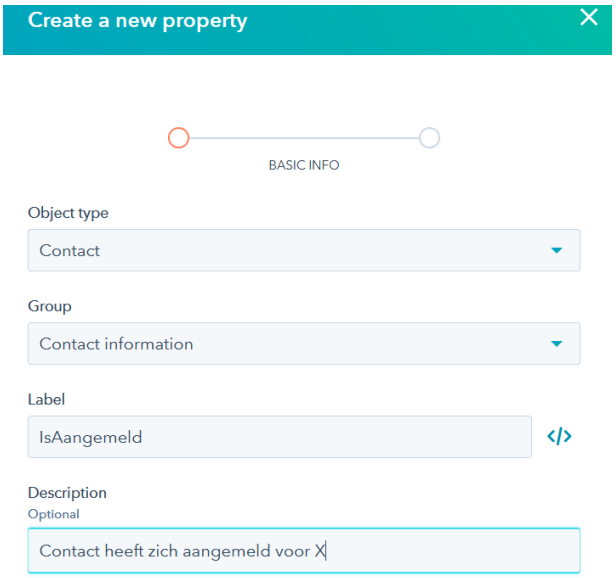
Aan de “crmobjects” tabel zit de “associations” tabel gekoppeld. Deze bevat twee ID’s van twee CRM objecten, die dus de koppelingen tussen de verschillende objecten zullen bijhouden. Een koppeling is bijvoorbeeld een contact die een bepaalde ticket heeft ingeschoten, of een deal die betrekking heeft tot een bepaalde line-item/product.

Ten slotte zijn er de “pipelines” en de “pipeline_stages” tabellen. Deze zijn er om de deals en tickets pipelines in bij te houden. Deze objecten verschillen dermate van de CRM objecten dat ze een eigen tabel nodig hadden. Een pipeline heeft meerdere stages.

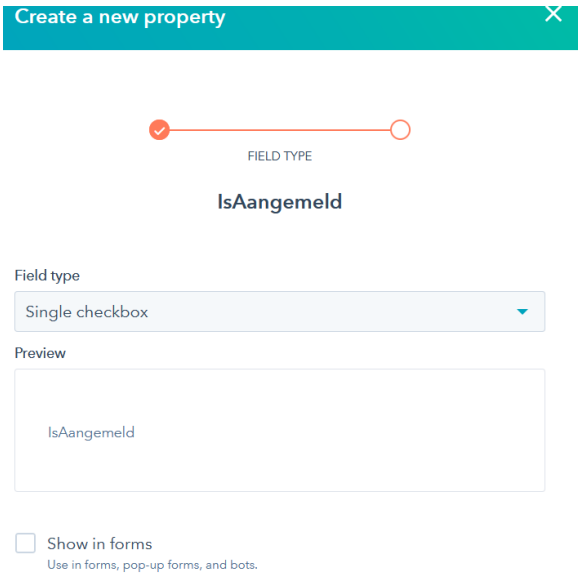
Dit zijn in eerste instantie alle tabellen die nodig zijn om een synchronisatie uit te voeren. Data die nodig is uit HubSpot, zoals informatie over de Rate Limits, zijn eenvoudig op te halen via een API en hoeven dus niet opgeslagen te worden.

3.4.2.2 Variabele datastructuur

Een van de complexiteiten die ontstaat bij het uitvoeren van een synchronisatie is het feit dat veel data objecten binnen HubSpot variabel zijn, in de zin dat een gebruiker zelf kan bepalen welke datavelden hij in elk object wil hebben. Deze worden bij HubSpot “Properties” genoemd. Bij een contact object zijn er rond de 100 properties beschikbaar binnen HubSpot die aan of uit kunnen worden gezet. Zo’n veld is bijvoorbeeld een voornaam, achternaam, straatnaam etc. Naast alle standaard datavelden is het voor een gebruiker ook mogelijk om zelf zo’n dataveld aan te maken, waarbij de gebruiker zelf de keuze heeft om te kiezen wat daar precies in kan komen te staan. Dit kan bijvoorbeeld een simpel invoerveld zijn, een datum of een checkbox. Hieronder is te zien hoe zo’n property kan worden aangemaakt.



Figuur 6: Aanmaken van een variabel dataveld in HubSpot, stap 1



Figuur 7: Aanmaken van een variabel dataveld in HubSpot, stap 2

Het is dus de bedoeling dat deze properties eerst worden gesynchroniseerd voordat de inhoudelijke data wordt gesynchroniseerd. Anders kan het bijvoorbeeld zijn dat je data wil invullen in een veld dat nog niet bestaat. Het grootste probleem hier is dat HubSpot zelf niet alle ingevulde velden terugstuurd bij een object. Als je een lijst van contacten ophaalt zonder enige headers stuurt HubSpot maar een paar specifieke velden terug. Bij een contact is dat bijvoorbeeld alleen een naam en emailadres, omdat dit verplichte velden zijn. Als je alle ingevulde velden wilt weten moet je deze specifiek op gaan vragen d.m.v. het toevoegen van een “properties” query parameter. Bij deze parameter moet je specifiek aangeven welke velden je extra wilt opvragen bij elk object.

Het probleem wat hierbij ontstaat is dat je van tevoren nog niet precies weet welke properties bij welk object actief zijn op de pagina en zijn ingevuld. Ook is het dus mogelijk om variabele properties te hebben, en deze wil je uiteraard ook graag in kunnen vullen bij de synchronisatie. Aangezien er ook enorm veel van deze properties zijn is het ook niet te doen om ze allemaal op te gaan vragen. Deze worden namelijk als URL query argument meegestuurd, en dus worden URL’s te lang als je er veel van opstuurt. Daarbij krijg je dan ook enorme objecten terug met heel veel overbodige data. Er zijn een aantal oplossingen om dit probleem op te lossen. Allereerst kan je er voor zorgen dat je per klant/omgeving in de synchronisatietool de nodige velden moet kunnen configureren. Dit is uiteindelijk ook nodig als je het mogelijk wil kunnen maken om de anonimisering van losse velden te kunnen configureren per klant. Die functionaliteit is uiteindelijk wel wenselijk om in zekere vorm te hebben, maar is in eerste instantie nog iets te complex om te implementeren.

Een eenvoudigere oplossing die dit probleem automatisch (deels) kan oplossen is om van te voren de tool te laten bepalen welke properties actief lijken te zijn, door te kijken naar de data die is opgehaald bij het synchroniseren van de properties. Er zijn namelijk een aantal velden in dit properties-object (te zien in figuur 5) die kunnen aanduiden wat voor soort property het is. Een van die velden is bijvoorbeeld “calculated”, die aangeeft of een veld wordt berekend vanuit bijvoorbeeld een andere variabele. Dit zegt dat HubSpot dit veld zelf invult, en daar hoeft dus niks aan te gebeuren vanuit de synchronisatietool. Een andere

variabele is bijvoorbeeld “hubspotDefined”, wat letterlijk zegt of HubSpot de waarde in die property zelf bepaald of dat deze handmatig moet worden ingevuld. Ten slotte bevat “modificationMetadata” een waarde “readOnlyValue”, die dus aangeeft of de waarde überhaupt ingevuld kan/mag worden door een gebruiker. Door de properties te filteren op deze specifieke waarde blijven er een handvol properties over die waarschijnlijk door een gebruiker/API kunnen worden ingevuld.

3.5 Configuraties

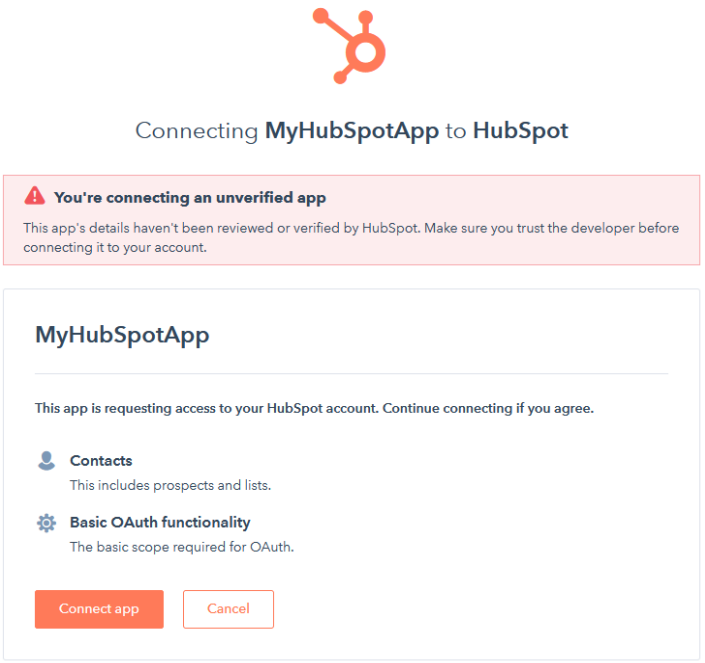
3.5.1 Aanleiding

De onderzoeksvraag die bij dit onderzoek hoort is: “Hoe kan de configuratie van klanten zo eenvoudig mogelijk opgezet worden zodat er geen code wijzigingen in de tool hoeven te gebeuren?”. Dit onderzoek was nodig om er achter te komen welke informatie er precies nodig is van een klant en van de verschillende accounts om een synchronisatie te kunnen starten. Daarbij moest nagedacht worden over hoe het ophalen van OAuth tokens werkt en hoe dat het best geïntegreerd kon worden in de applicatie. Deze tokens moesten ook ergens veilig worden opgeslagen, dus daar moest ook naar gekeken worden. Ten slotte moest worden nagedacht over welke ander configuraties eventueel mogelijk moeten/zouden moeten zijn, zoals het aan- en uitzetten van specifieke synchronisaties.

3.5.2 Resultaat

Voor het starten van de synchronisatie moet de software weten uit uit welk HubSpot account hij zijn data moet halen en naar welk HubSpot account deze data heen moet. Daarvoor moet er dus eerst gezorgd worden dat de klanten met de nodige accounts (productie en developers) worden geconfigureerd.

Er zijn twee manieren om API’s aan toe roepen bij een specifiek account in HubSpot: via een OAuth verbinding, of zonder OAuth met alleen een API key. Als een gebruiker OAuth niet heeft ingeschakeld bij zijn account, dan kan er bij een API een API key worden meegestuurd die HubSpot verteld bij welk account hij de query moet uitvoeren. Als er wel OAuth op zit, dan moet ontwikkelaar die de API wil aanroepen eerst zijn applicatie binnen het developers platform van HubSpot registreren. Vervolgens kan de gebruiker waar het HubSpot account van is zijn account koppelen aan die applicatie via een specifieke URL. Zodra dit is gebeurd krijgt de applicatie een specifieke code terug, waar een OAuth Access Token & Refresh Token mee opgehaald kunnen worden. Deze moeten dan gebruikt worden bij het aanroepen van de API, en dan weet HubSpot gelijk over welk account het gaat. De API key is daarbij dus niet nodig. Hieronder is een screenshot te zien van het verbinden van een account met een HubSpot-geregistreerde applicatie.



Figuur 8: Gebruiker geeft toegang aan applicatie om API's aan te kunnen roepen middels OAuth

Een andere configuratie die eventueel nodig kan zijn is de mogelijkheid om specifieke taken van de synchronisatie in/uit te schakelen. Het kan bijvoorbeeld zo zijn dat je niet alle data wilt synchroniseren, of dat je niet alles wilt anonimiseren. Dit zou een gebruiker van de synchronisatietool specifiek kunnen aangeven bij het starten van de synchronisatie. Deze functionaliteit heeft echter niet de hoogste prioriteit voor de ontwikkelaars bij CODE14, en er zal dus pas na het implementeren van de hoofd-functionaliteiten naar gekeken worden..

3.6 Functioneel ontwerp

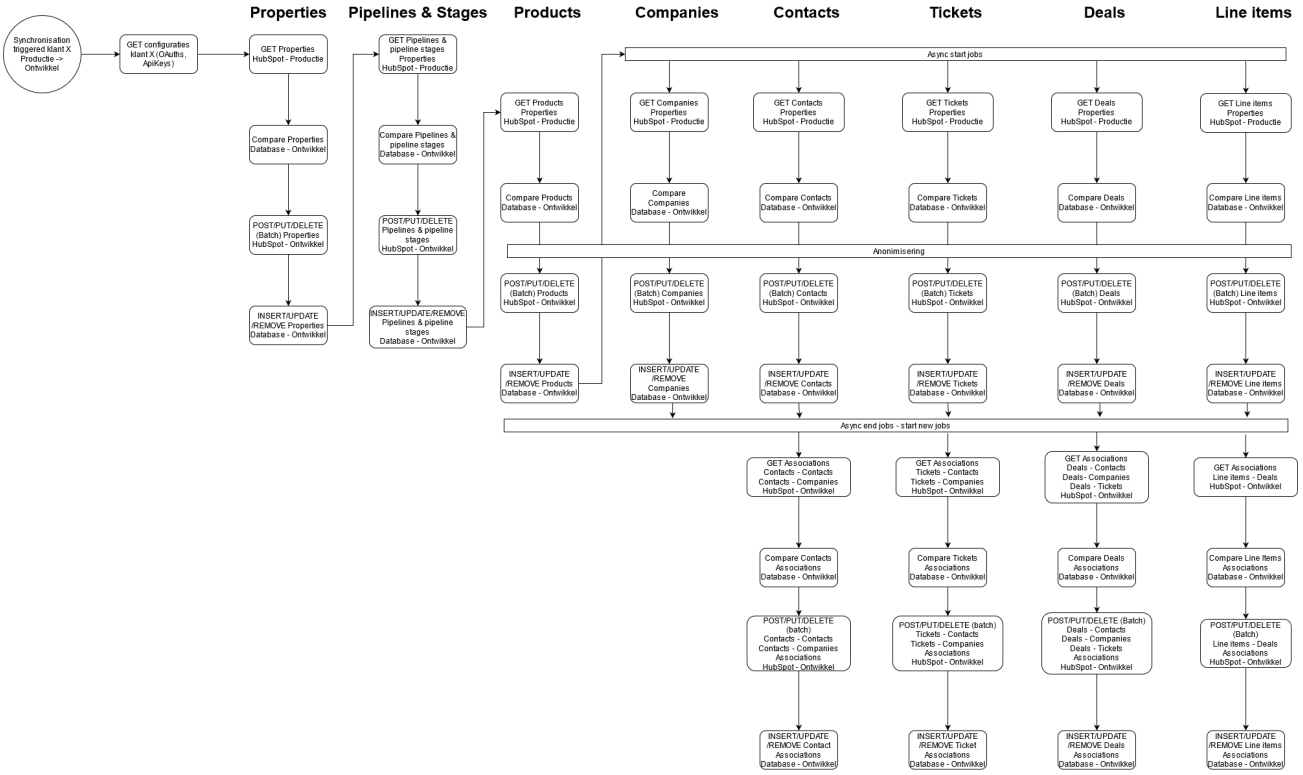
3.6.1 Synchronisatie

De synchronisatie bestaat uit een aantal verschillende onderdelen, die in bepaalde volgorde moeten worden gedraaid:

- Properties
- Pipelines (+ stages)
- Contacts
- Companies
- Tickets
- Products
- Line Items

- Deals

De volgorde hiervan is belangrijk omdat een aantal objecten afhankelijk zijn van andere om data goed te kunnen synchroniseren. Pipelines zijn bijvoorbeeld de trajecten die een deals/ticket kunnen doorlopen, en bestaan uit verschillende stages (bij een deal bijv. presentatie gepland, offerte verstuurd etc.). Het is dus belangrijk dat die pipelines zijn gesynchroniseerd voordat de tickets/deals worden gesynchroniseerd. Het kan dan namelijk zijn dat er een nieuwe pipeline is aangemaakt. De deals die daar dan eventueel in staan kunnen dan dus niet (goed) worden gesynchroniseerd. Daarnaast hebben objecten als contacts en companies vaak onderlinge koppelingen genaamd “Associations”. Deze kunnen ook pas worden gesynchroniseerd als alle losse dataobjecten zijn ingevuld. Daarbij zijn er ook een aantal objecten die niet op elkaar hoeven te wachten en dus tegelijkertijd kunnen draaien. Hieronder in figuur 9 is de gehele flow te zien van de volgorde van synchronisaties:



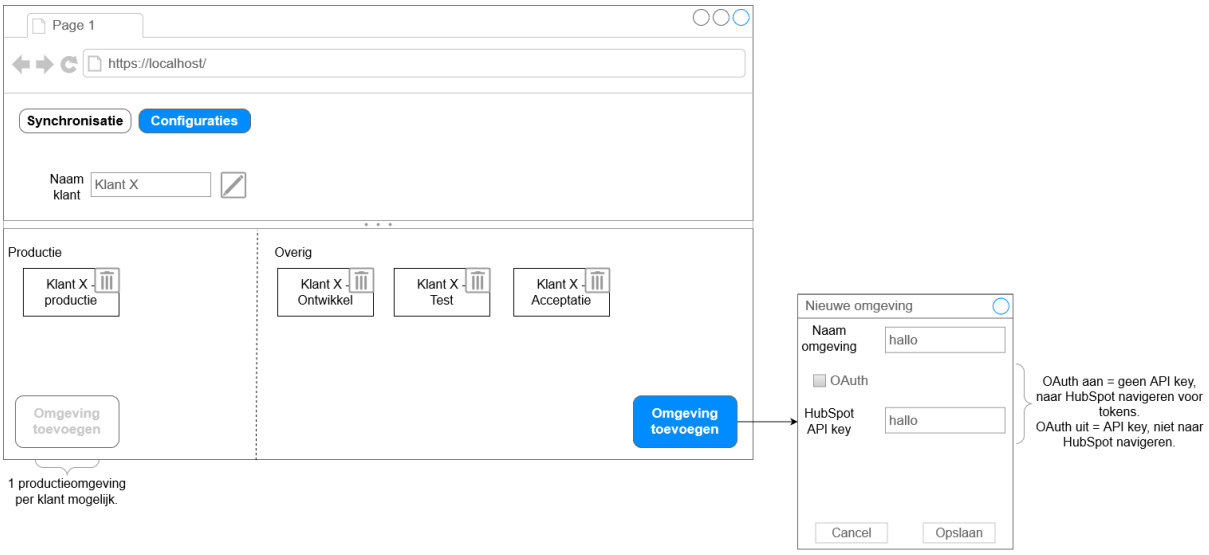
Figuur 9: Flowchart van de volledige synchronisatie

De flow begint linksboven, met de trigger van de synchronisatie van klant X, omgeving productie naar omgeving ontwikkel. Van beide omgevingen moeten vervolgens de nodige configuraties opgehaald worden, zoals OAuth tokens, API Keys etc. Dan is de eerste stap in de synchronisatie het synchroniseren van de Properties. De properties worden opgehaald bij HubSpot en worden vergeleken met de database om erachter te komen welke van de properties zijn gewijzigd sinds de vorige synchronisatie (als die er niet is geweest zijn in principe alle properties “wijzigingen”). Vervolgens worden dus de wijzigingen verstuurd naar HubSpot via API calls (POST’s, PATCH’s, DELETE’s). De responses van calls worden dan ingevuld in de database, zodat deze de volgende synchronisatie gebruikt kunnen worden om nieuwe wijzigingen te herkennen.

Dit proces van ophalen uit HubSpot, vergelijken met de database, opsturen naar HubSpot en invullen in de database wordt na de properties herhaalt bij de Pipelines en stages, en vervolgens bij de Products. Daarna starten een aantal jobs tegelijkertijd, die ook deze zelfde flow aanhouden, bij de Companies, Contacts, Tickets, Deals en Line Items. Als deze allemaal zijn geweest is het alleen nog de beurt aan de associations. In de flowchart staat beschreven welke associations er zijn tussen de verschillende objecten (contact - contact, contact - companies etc.). Qua synchronisatie volgen deze nog wel dezelfde flow als een normaal object.

3.6.2 UI/Front-end

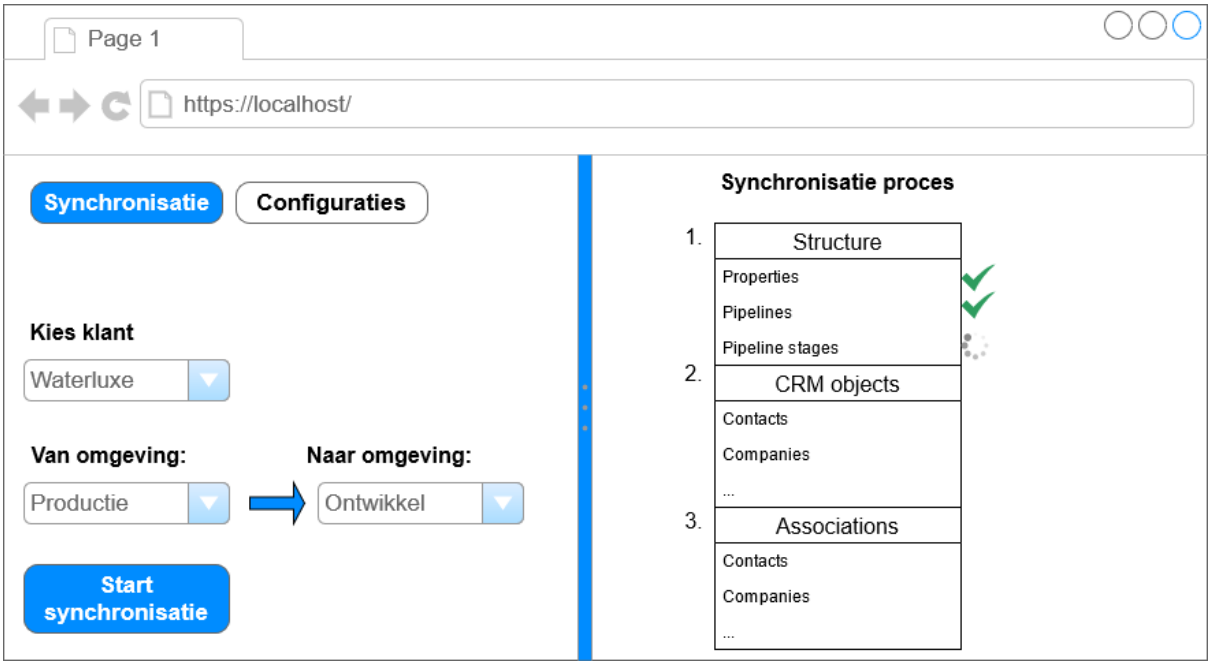
Binnen de UI moeten een aantal dingen kunnen gebeuren. Allereerst moet het mogelijk zijn om klanten te kunnen registreren. Hier is niet veel informatie voor nodig, voornamelijk alleen een naam en een ID. Binnen een klant moeten dan (HubSpot) accounts kunnen worden gesynchroniseerd. Deze moeten worden gecategoriseerd in productie en ontwikkelings-accounts. Zo blijft het namelijk duidelijk welk account echt van de klant is en welke gebruikt worden voor ontwikkelingen binnen CODE14. Een account moet ook bestaan uit een naam, en er moet aangegeven worden of het account beveiligd is met OAuth of niet. Zo nee, kan er gewoon een API key uit HubSpot worden ingevuld en is dit genoeg. Zo ja, moet de gebruiker worden doorgeschakeld naar HubSpot om zijn account te koppelen aan de synchronisatietool-applicatie. De synchronisatietool krijgt dan een code terug om OAuth tokens mee op te halen. Hieronder in figuur 10 is een wireframe te zien van deze account-pagina, zoals hij er ongeveer uit moet gaan zien:



Figuur 10: Wireframe 1: detailpagina clients --> accounts

Als er vervolgens een klant is geregistreerd met in elk geval twee accounts (productie en ontwikkel) moet een synchronisatie kunnen worden gestart. Er moet dan een optie zijn om een “van” en “naar” account te kiezen, en om de synchronisatie te kunnen starten. Het “naar” account mag nooit het productie account zijn. Zodra de synchronisatie is aangezet moet het binnen de UI mogelijk zijn om (globaal) te kunnen zien hoe ver de synchronisatie is, of er fouten zijn opgetreden etc. Hieronder in figuur 11 is globaal te zien hoe dit er

ongeveer uit gaat zien. Het kan zijn dat de resultaten naar een aparte pagina worden geplaatst.



Figuur 11: Wireframe 2: synchronisatie is gestart van productie naar ontwikkel

3.7 Requirements

3.7.1 Inleiding

Hieronder zijn de requirements te vinden die aangeven waar het project allemaal aan moet/zou moeten voldoen. Deze zijn opgesteld na het uitvoeren van het onderzoek, maar een aantal zaken waren al wel eerder bekend dat het requirements zouden worden. Op basis van deze requirements zijn uiteindelijk de ontwerpen, en later ook de User Stories opgesteld. Een deel van de requirements stonden al vast op basis van de opdrachtgeving en werkwijze binnen CODE14, zoals het waarborgen van kwaliteit en een aantal functionaliteiten van de tool. Andere stonden nog deels open voor invulling, zoals welke frameworks gebruikt gaan worden.

De requirements zijn opgedeeld in 3 categorieën: user, business en systeem requirements. User requirements zijn zaken wat een gebruiker zou moeten kunnen doen in of met de applicatie. Business requirements geven aan hoe de applicatie ingezet moet worden, en hoe het de business vooruit kan helpen. Systeem requirements geven aan waar de applicatie technisch aan moet voldoen, en wat de backend bijvoorbeeld moet kunnen afhandelen. Daarbij hebben ze een functionele/niet functionele categorisering gekregen, en een prioriteit score.

3.7.1 User

Nummer	Beschrijving	Functioneel/ niet functioneel	Prioriteit
U1	Een gebruiker moet zonder code wijzigingen een klant kunnen registreren/configureren.	F	Must
U2	Een gebruiker moet eenvoudig een synchronisatie kunnen starten voor een klant naar een omgeving naar keuze.	F	Must
U3	Een gebruiker moet een synchronisatie via een UI gestart kunnen worden.	F	Must
U4	Een gebruiker moet een synchronisatie via een externe API kunnen starten.	F	Should
U5	Een gebruiker moet een synchronisatie via de commandline kunnen starten.	F	Should
U6	Een gebruiker moet specifiek kunnen aangeven welke data hij gesynchroniseerd wil hebben.	F	Could
U7	Een gebruiker moet de anonimisering kunnen configureren naar eigen wensen.	F	Could
U8	Een gebruiker moet resultaten van de synchronisatie (logs) altijd kunnen opvragen	F	Should

3.7.2 Business

Nummer	Beschrijving	Functioneel/ niet functioneel	Prioriteit
B1	De synchronisatietool moet voor elke HubSpot klant inzetbaar kunnen zijn.	F	Must
B2	Het HubSpot pakket dat een klant heeft zou geen invloed moeten hebben op de werking	F	Must

	van de synchronisatietool.		
B3	De synchronisatie moet de huidige situatie van handmatig data invullen in HubSpot verbeteren.	NF	Must
B4	De kwaliteit van de projecten moet worden gewaarborgd door de QA eisen van CODE14	NF	Must
B5	De code moet eenvoudig leesbaar zijn.	NF	Must
B6	De code moet zo zijn opgesteld dat bij het toevoegen van nieuwe synchronisatietaken er geen extra werk nodig is aan de bestaande synchronisatietaken.	NF	Should
B7	De front-end moet afgeschermd worden met een login systeem	F	Should

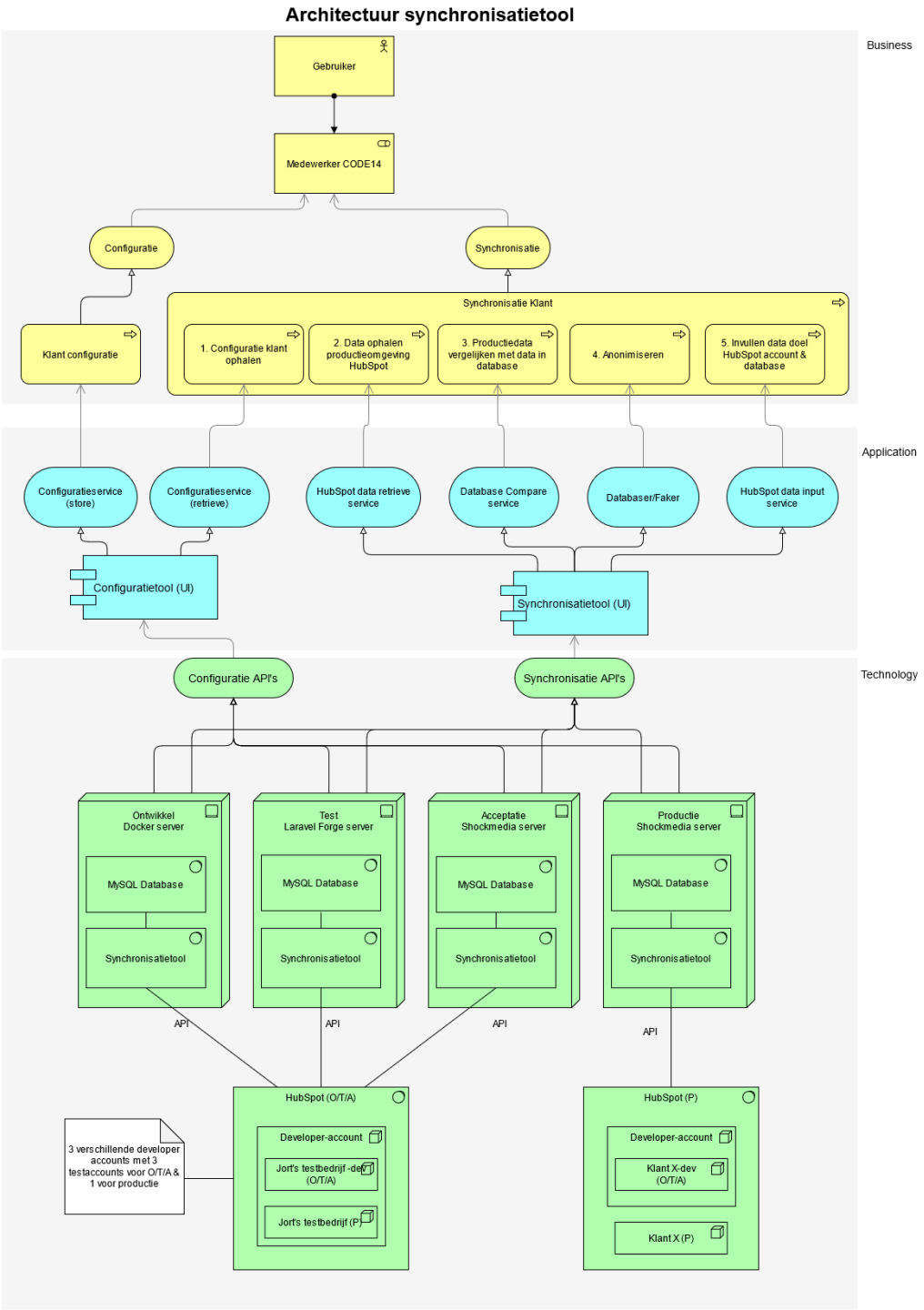
3.7.3 Systeem

Nummer	**Beschrijving**	**Functioneel/ niet functioneel**	**Prioriteit**
S1	Als back-end framework moet Laravel PHP worden gebruikt.	NF	Must
S2	Als front-end framework moet Vue JS worden gebruikt.	NF	Should
S3	De ontwikkelomgeving van de tool moet in Docker kunnen worden gehost.	NF	Must
S4	De testomgeving van de tool moet in Laravel Forge kunnen worden gehost.	NF	Must
S5	De acceptatieomgeving van de tool moet in Shockmedia kunnen worden gehost.	NF	Must
S6	De productieomgeving van de tool moet in Shockmedia kunnen worden gehost.	NF	Must

S7	Als database moet MySQL gebruikt worden.	NF	Must
S8	Klant-configuraties moeten AES encrypted worden opgeslagen met een key uit de omgevingsvariabelen.	F	Must
S9	Verschillende processen van de synchronisatie moeten asynchroon kunnen draaien.	NF	Should
S10	De voortgang van de synchronisatie moet live zichtbaar gemaakt kunnen worden in de front-end.	NF	Should
S11	Er moeten voor verschillende functionaliteiten van de tool tests worden geschreven om deze te valideren.	NF	Should
S12	De tool moet zelf bepaalde velden kunnen valideren om fouten in HubSpot te voorkomen (bijv email formaat).	F	Should
S13	De synchronisaties moeten opgezet worden als een soort pipeline: nieuwe synchronisaties worden een soort job en komen in een queue tot er een plek vrij is om hem uit te voeren.	NF	Could
S14	Resultaten van de synchronisatie moeten worden gelogd	F	Should

4 Systeembeschrijving

4.1 Architectuur



Figuur 12: Architectuurplaat synchronisatietool

Hierboven is de architectuurplaat te zien van de synchronisatietool, verdeeld in een businesslaag, een applicatielaag en een technologielaag. In de businesslaag zijn de twee

processen te zien van het registreren van een klantomgeving en het uitvoeren van een synchronisatie. Deze worden normaal gesproken uitgevoerd door een medewerker van CODE14. In de applicatielaag is een scheiding gemaakt tussen de synchronisatietool en de configuratietool. In feite is dit beschikbaar binnen één project, maar om de twee processen gescheiden te houden zijn deze uit elkaar gesplitst. Daarbij zijn de processen gekoppeld aan een service die beschrijft wat er mogelijk is binnen de applicatie. De communicatie tussen de UI en de back-end synchronisatietool loopt via API's. De synchronisatietool praat vervolgens dan tegen een MySQL database en tegen HubSpot aan: de front-end heeft geen rechtstreekse koppeling met de database en ook nooit met HubSpot (enkel een redirect naar HubSpot). De MySQL database bevat alle eerdere synchronisaties van alle klanten & accounts. Binnen HubSpot zijn de losse accounts van een klant/ontwikkelaar te zien. Jort's testbedrijf zijn accounts die worden gebruikt om de synchronisatie te kunnen testen, Klant X (P) is een account die een klant daadwerkelijk gebruikt, en Klant X-dev zijn accounts die een ontwikkelaar bij CODE14 gebruikt om bijvoorbeeld koppelingen te ontwikkelen/testen. De naam die deze accounts dragen (O/T/A/P) zijn er alleen om aan te geven voor welke omgeving van bijvoorbeeld een koppeling van CODE14 deze bedoeld zijn. Het zijn dus aparte accounts, maar bevinden zich niet in een andere soort HubSpot versie o.i.d., behalve dat O/T/A HubSpot test-accounts zijn, en P normale HubSpot accounts.

4.2 Front-end

4.2.1 Vue

De front-end van de synchronisatietool is gebouwd in het framework "Vue.js". Vue is een Javascript front-end framework, voornamelijk bedoeld voor het bouwen van SPA's (Single Page Application). Aangezien de front-end van de tool relatief klein is en er niet heel veel pagina's zijn was er ook de keuze om niet voor een los framework te gaan, maar bijvoorbeeld voor de ingebouwde front-end mogelijkheden van Laravel genaamd Blade. Uiteindelijk is de keuze toch gegaan om de front-end van de applicatie in Vue te bouwen omdat dit toch iets meer mogelijkheden biedt dan Blade op het gebied van bijvoorbeeld componenten, en omdat het de code voor de front-end een stuk overzichtelijker maakt. Vue bestanden zijn namelijk een stuk duidelijker ingedeeld dan Blade bestanden, en het verwerken van data uit de back-end gaat een stuk eenvoudiger. Daarbij had ik zelf nog niet veel ervaring met het Vue framework, dus ging er ook voorkeur uit om dit een keer te leren.

4.2.2 TailwindCSS

Om tijd te besparen met het ontwerpen van een mooie front-end interface is er gebruik gemaakt van een CSS framework, namelijk Tailwind. Er is voor Tailwind gekozen omdat dit een relatief lichtgewicht framework is tegenover bijvoorbeeld een component library als BootstrapVue of Vuetify. Het gebruiken van een grotere component library was makkelijker geweest om sneller een mooie front-end in elkaar te zetten, maar was uiteindelijk iets te overbodig gezien de grootte van de front-end zelf. Om toch te voorkomen dat alle CSS zelf geschreven moest worden is er dus uiteindelijk voor gekozen om TailwindCSS te gebruiken, aangezien dit een mooie, duidelijke styling biedt en ook voorbeelden aanbiedt op hun website. Ten slotte verwijderd TailwindCSS bij een build voor productie alle styling classes die niet worden gebruikt, waardoor het extra klein is als het aan een client wordt geserveerd.

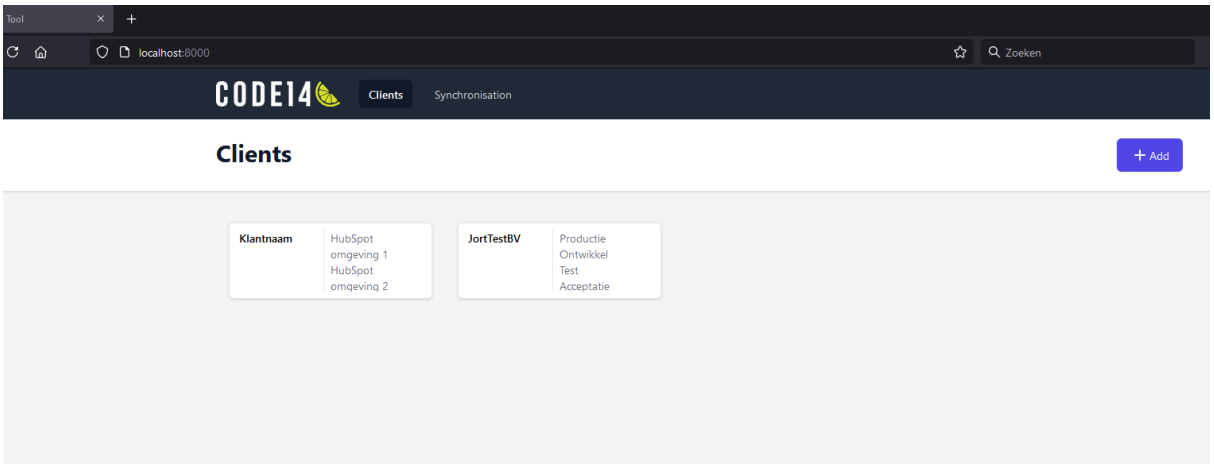
4.2.3 Webpack

Voor het bundelen van de applicatie wordt gebruik gemaakt van Webpack. Webpack zorgt ervoor dat alle losse bestanden die behoren tot de front-end worden samengevoegd tot een aantal statische assets: alle Javascript en HTML in Vue bestanden worden samengevoegd tot een enkel Javascript bestand en alle styling wordt samengevoegd tot een enkel CSS bestand. Deze komen vervolgens, samen met alle afbeeldingen, in een specifieke map te staan die vervolgens wordt gebruikt om de front-end te serveren. Het gebruik van een bundler is nodig omdat een ge-build project anders niet weet welke bestanden hij aan de client moet serveren. Deze horen namelijk in een specifieke map te staan (public), wat het startpunt is voor het project.

4.2.4 Werking

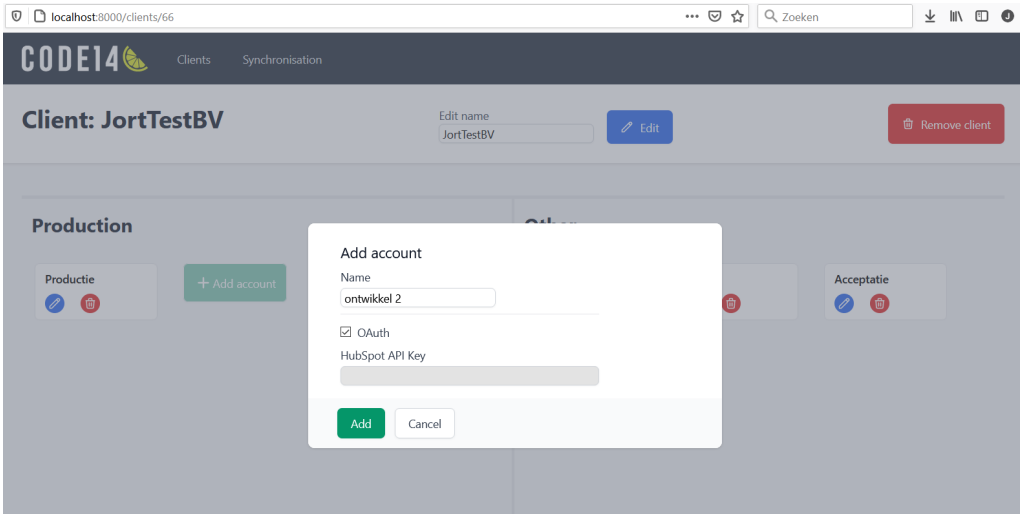
De flow van de tool is als volgt:

Een gebruiker navigeert naar de url van de synchronisatietool. Dit kan een lokale versie zijn dat een ontwikkelaar draait, of (in de toekomst) een versie op een centrale server die voor alle ontwikkelaars bij CODE14 beschikbaar is. Dit is echter nog niet het geval, en daarvoor zal ook een inlogsysteem geïmplementeerd moeten worden. Hier wordt dan eerst een pagina weergegeven met een overzicht van klanten. Er kan een klant worden aangemaakt, of er kan er eentje worden geselecteerd om naar de detailpagina te gaan:



Figuur 13: Screenshot van klant overzichtspagina synchronisatietool

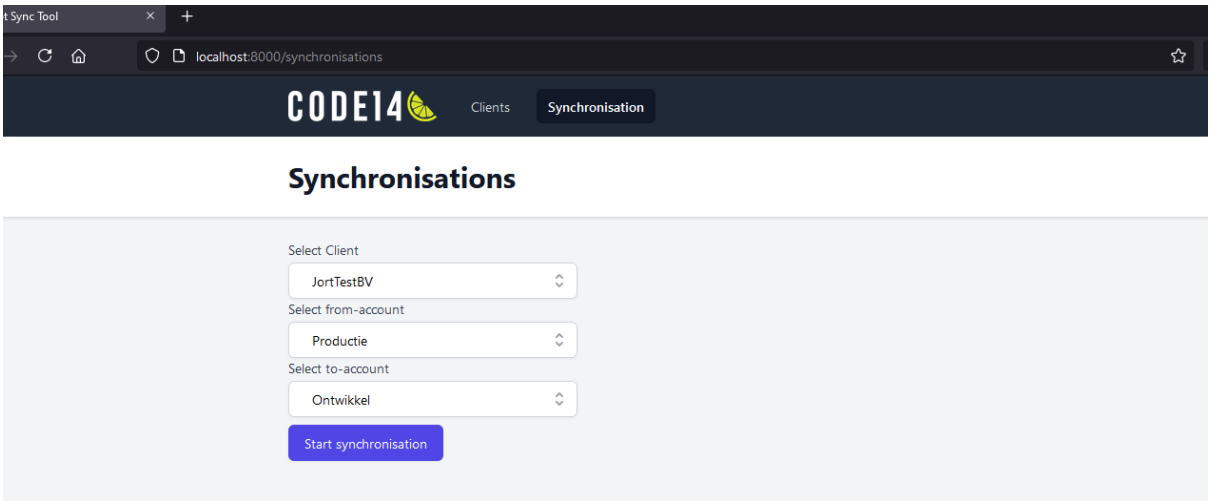
Bij het selecteren van een klant wordt de gebruiker doorgestuurd naar de detailpagina. Hier kan de naam van de klant worden gewijzigd, kan de klant worden verwijderd en kunnen accounts worden toegevoegd, gewijzigd en verwijderd. Hieronder is deze pagina te zien, waarbij de pop-up voor het toevoegen van een account is geopend:



Figuur 14: Screenshot van aanmaken account op detailpagina klant

Bij het selecteren van “OAuth” geeft een gebruiker aan dat dit account is afgeschermd met OAuth tokens. Hiervoor is een redirect naar HubSpot nodig, waarbij de gebruiker moet aangeven dat de app (de synchronisatietool) toegang mag hebben tot de API’s van dat account. Bij terugkomst wordt het account aangemaakt. De goedkeuring voor het aanroepen van de API’s is te zien in figuur 8.

Zodra verschillende accounts zijn aangemaakt kan worden door genavigeerd naar de synchronisatie pagina. Hier kan een klant worden geselecteerd, en vervolgens ook een “vanuit” en een “naartoe” account, die aangeven waar data uitgehaald moet worden en waar deze naartoe moet. Vervolgens kan met “start” de synchronisatie worden gestart. Deze pagina is hieronder te zien in figuur 15:



Figuur 15: Screenshot van starten synchronisatie

Na het drukken op start wordt de synchronisatie in de backend gestart. In de huidige implementatie gebeurd die synchronisatie alleen nog in de backend zonder dat er feedback wordt gegeven op de voortgang. Uiteindelijk zal het doel zijn om in de frontend een feedback tabel weer te geven die aangeeft hoe ver de synchronisatie is en of alles geslaagd is.

4.2.5 Projectstructuur

De front-end bestaat uit verschillende soorten bestanden, die elk een andere functie hebben. Aan het begin staat het bestand “App.vue”, wat dient als startpunt voor de front-end. Deze bevat vrijwel alleen het component “RouterView”: een component die een specifiek ander component kan renderen op basis van de route. Dit zorgt er dus voor dat de applicatie een SPA applicatie is: in plaats van een gebruiker door te sturen naar een specifieke URL wordt een bepaalde route aan de Vue Router toegevoegd. Deze render dus alleen een andere content in de pagina, in plaats van de volledige pagina te herladen.

De pagina’s/componenten die worden gerenderd door de RouterView zijn dus de individuele pagina’s binnen het project. Dit worden hier de “views” genoemd. Een view is dus bijvoorbeeld de klanten overzichtspagina, de klanten detailpagina of de synchronisatie pagina.

Bij het inladen van een pagina moet vrijwel altijd data worden opgehaald vanuit de back-end. Hiervoor zijn voor elke Controller in de back-end “services” aangemaakt (AccountsService, ClientsService etc.). Deze services bevatten implementaties voor het aanroepen van de back-end API’s, en sturen de resultaten terug naar de views.

Naast views bestaan er ook Vue componenten. Dit zijn front-end componenten die binnen een pagina bestaan, maar niet vast op de pagina horen of door meerdere pagina’s worden gebruikt. Denk aan Modals/Pop Ups, de navigatiebalk etc.

4.3 Back-end

4.3.1 Laravel

Voor de back-end wordt het framework Laravel (PHP) gebruikt. Laravel is een Open Source PHP web-framework die voornamelijk back-end gericht is, maar full-stack gebruikt kan worden. Laravel onderscheid zich door veelgebruikte taken in web-development makkelijker te maken en “magisch” op te lossen. Een van de beste voorbeelden hiervoor is “Eloquent”. Eloquent is een object-relatie mapper binnen Laravel die interactie met de database makkelijker maakt. Een voorbeeld waar dit je helpt is bijvoorbeeld bij een “Get By ID” API in een Controller. Normaal gesproken zou je een methode bouwen die gedefinieerd wordt als API endpoint, waarin je een ID verwacht als parameter. Met deze ID ga je vervolgens naar de database en ga je kijken of er een model in de database staat die dat ID heeft, en als dat niet zo is stuur je een foutmelding terug. Door het gebruiken van Eloquent kan in de parameters van die methode al definiëren welk model je graag wil hebben in die methode. Een gebruiker van de API kan deze API gewoon aanroepen met een ID als parameter, en Laravel kan dan onder water al precies het ophalen van dat object/model voor jou doen, en stuurt ook zelf al een foutmelding terug als er niets gevonden is.

Dit soort “magische” trucjes zitten door heel Laravel heen, en is ook een van de voornaamste redenen waarom het bij CODE14 bijna altijd als voorkeur gebruikt wordt. Omdat het niet nodig was om een specifieke taal/framework te gebruiken voor mijn probleem en omdat ik zelf ook graag Laravel & PHP een keer goed wou leren ben ik dit ook gaan gebruiken in mijn project.

4.3.2 Faker

Faker is de package die zorgt voor de anonimisering van de data tijdens de synchronisatie. De functionaliteit van de Faker is erg simpel: het genereert fake data voor allerlei verschillende doeleinden. Het heeft toegang tot een enorme dataset met verschillende soorten data (namen, adressen, domeinnamen, foto's etc.), en bevat ook logica om verschillende soorten data te genereren (IP-adressen, datums, barcodes etc.). Ten slotte bevat het voor verschillende landen, waaronder Nederland, een aantal lokaal-specifieke generaties. Voor Nederland is dat bijvoorbeeld een functietitel of een IBAN nummer.

De specifieke Faker die is gebruikt in de applicatie is FakerPHP. Dit is een van de grootste Fakers voor PHP, en wordt nog constant geüpdatet voor o.a. nieuwe PHP versies. De meest gebruikte Faker voor PHP is daar namelijk mee gestopt, dus om het project iets meer future-proof te maken is FakerPHP gekozen.

Het gebruik van de Faker is erg eenvoudig. Nadat de package is geïnstalleerd instantieer je een Faker object met behulp van een Faker "Factory". Vervolgens kun je direct via dit object al je data genereren, door bijvoorbeeld *$faker->firstname* aan te roepen. In eerste instantie worden maar een aantal standaard properties hardcoded ge-faked, waarvan duidelijk is dat dit invulbare data is in HubSpot (namen, adressen, mailadressen etc.). Binnen de synchronisatie gebeurd deze anonimisering/datagenerering tussen het ophalen van data en het opsturen naar HubSpot. De flow is dus meestal alsvolgt:

1. GET data object uit HubSpot
    a. Als er al een synchronisatie is gedaan: haal alleen de nieuwste/gewijzigde data op. Zoals eerder is beschreven in kop 3.4 kan dit met behulp van de "Search" API, waarin een filter kan worden toegevoegd die alleen data ophaalt waarbij de wijzigingsdatum groter/nieuwer is dan een bepaalde timestamp.
2. Transform/map resultaat naar een input object voor HubSpot
    a. Anonimiseer de nodige velden met de Faker
3. POST data object naar HubSpot
4. Transform/map resultaat van de POST naar een CRMObject model voor de database. Daar wordt dan de ID van het originele object uit de productie omgeving aan toegevoegd.
5. Sla het CRMObject op in de database

4.3.3 HubSpot

Voor de interactie met HubSpot wordt gebruik gemaakt van de HubSpot API Client. Dit is een package gemaakt door HubSpot zelf, en is beschikbaar voor verschillende talen (Node.js, PHP, Ruby, Python). De package bevat alle API's die beschikbaar zijn in HubSpot, duidelijk gestructureerd en eenvoudig aan te roepen. De client kan worden aangemaakt met een API key of Access Token die dan bij elke API wordt toegevoegd, en alle nodige parameters zijn mee te geven aan de client.

Figuur 16: Code snippet van het gebruiken van de HubSpot Client en het ophalen contacts

Hierboven in figuur 16 is een kleine code-snippet te zien waarin de HubSpot client wordt gebruikt. Op de eerste regel (61) wordt een HubSpot client aangemaakt met een API key. Vervolgens worden een aantal variabelen aangemaakt en wordt een while-loop geopend. Het doel van deze code is om alle contacten op te halen uit het account waar de API key bij hoort. In dit geval doen we dat met maar 1 contact per keer. Regel 68 laat zien hoe met behulp van de client een API aangeroepen kan worden: contacten vallen onder het pakket CRM, en in dit geval willen we gewoon de basic API (andere API's zijn bijvoorbeeld search, batch of associations). Omdat we een pagina van contacten willen ophalen roepen we de getPage() methode aan. Naast getPage kun je hier bijvoorbeeld ook contacten toevoegen, updaten en archiveren. Uit deze API komen twee objecten: de resultaten van de API en een "Paging" object. Dit paging object zit erbij om aan te geven welke page parameter gebruikt moet worden als je de volgende pagina wilt ophalen. Deze is gebaseerd op het ID van het laatste object uit de pagina die je hebt opgehaald, dus je wil de volgende pagina hebben "after" in dit geval eerst ID 0. Ook al bestaat er geen object met ID 0 kan HubSpot dit zelf afhandelen door het eerstvolgende object te pakken.

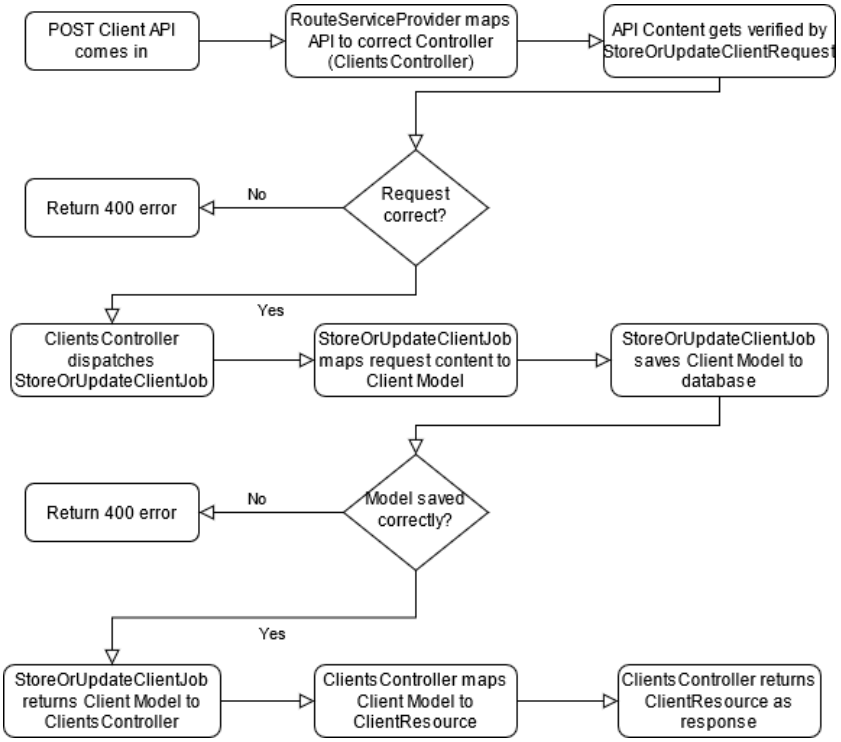
Voor het rekening houden met de rate limits is er een stukje middleware toegevoegd aan de client die de API calls afhandeld naar HubSpot. Deze middleware zorgt ervoor dat op het moment dat er een "429" code als response wordt gestuurd (Too Many Requests), de client 10 seconden wacht en vervolgens dezelfde request opnieuw verstuurd. In een ideale situatie is het eigenlijk de bedoeling dat deze code helemaal niet moet voorkomen, maar dat er tot maar de helft van de limiet calls worden gedaan. Dit bleek echter iets te lastig om goed te implementeren, dus is voor de eerste versie buiten wegen gelaten.

4.3.4 Projectstructuur

Een Laravel project bevat standaard allerlei verschillende mappen en bestanden. Een aantal van deze mappen/bestanden worden echter vrijwel nooit aangepast en bevatten bijvoorbeeld standaard configuraties voor Laravel (config). Het meeste eigen werk dat in de back-end wordt gedaan gebeurd in de "app" folder. Hier worden namelijk alle Controllers gedefinieerd en wordt alle dataverwerking gedaan. In mijn geval wordt hier dus ook de synchronisatie gedraaid. Andere interessante mappen zijn "database": hier staan alle

migrations en seeders in voor de database, en “resources”: hier staan alle nodige front-end bestanden in.

De standaard back-end flow is als volgt opgebouwd:



Figuur 17: Flow van het verwerken van een API call

Binnen de Http folder worden routes gedefiniëerd. Deze routes verwijzen binnenkomende API calls door naar de nodige Controller, die onder Http/Controllers worden aangemaakt. In dit geval is “ClientsController” een controller die API’s bevat voor CRUD opties voor client objecten. Als de API call bedoeld is om een nieuw model aan te maken (POST) wordt er een “Request” class aangemaakt. Deze request class bevat een aantal eisen waar de request aan moet voldoen, bijvoorbeeld dat bepaalde velden verplicht moeten zijn. In dit geval is dit de “StoreOrUpdateClientRequest”, die regels bevat voor het aanmaken of wijzigen van client objecten.

Als de request geverifieerd is d.m.v. de Request class trapt de controller een Job af. Een Job heeft vaak maar een enkele specifieke taak, bijvoorbeeld om een object aan te maken/ te updaten in de database, of om een lijst van objecten op te halen. In dit geval is er dus de “StoreOrUpdateClientJob”, die een client uit de database kan aanmaken of updaten als deze al bestaat. Bij het aanmaken van zo’n object wordt de request omgebouwd naar het verwachtte Model. Als dit goed is gegaan krijgt de controller het resultaat terug uit de Job. Als je bijvoorbeeld een object wilt aanmaken is dit resultaat het aangemaakte object. Dit object wordt gemapt naar een Resource class, die vervolgens terug wordt gestuurd als response naar de aanroeper van de API. Zo’n Resource class bevat dan alleen de velden die nodig zijn bij diegene die de API aanroept, in mijn geval is dit alleen de front-end. In de basis is dit hoe de structuur in de back-end in elkaar zit.

4.3.5 Synchronisatie

De synchronisatie wordt gestart via de UI van de synchronisatietool. Bij het openen van de pagina wordt een lijst met clients opgehaald, en deze worden weergegeven in een dropdown menu. Na het selecteren van een client worden ook de onderliggende accounts weergegeven. Vervolgens moet er een “from” en een “to” account worden gekozen, die aangeven waar de data moet worden opgehaald en waar die moet worden ingevuld. Zodra beide zijn geselecteerd stuurt de front-end een API request naar de back-end met beide accounts. Deze pagina is te zien in figuur 15. Op dit moment is het alleen nog mogelijk om de properties te synchroniseren, samen met de property groups, en om de contacten te synchroniseren. Het implementeren van de rest van de CRM objecten is echter geen grote klus omdat ze qua implementatie vrijwel hetzelfde zullen zijn als de contact-synchronisatie. Door gebrek aan tijd hebben deze echter de eindoplevering nog niet gehaald.

Eenmaal in de back-end wordt de synchronisatie gestart. Elk object die moet worden gesynchroniseerd (properties, pipelines, contacts etc.) heeft een eigen Job. Door het gebruik van een “Bus” kan de volgorde worden bepaald waarop de jobs moeten worden uitgevoerd. Omdat voor een aantal jobs de volgorde niet uit maakt en deze in principe parallel kunnen draaien worden deze in een “Chain” gestopt binnen de bus. Op deze manier bevat de bus dus een batch van chains, waarbij de volgende chain pas wordt uitgevoerd als alle jobs uit de voorgaande chain zijn afgerond. De volgorde is als volgt:

Chain 1:

- Properties

Chain 2:

- Pipelines
- Pipeline stages

Chain 3:

- Products

Chain 4:

- Companies (CRM-object)
- Contacts (CRM-object)
- Tickets (CRM-object)
- Deals (CRM-object)
- Line items (CRM-object)

Chain 5:

- Companies (Associations)
- Contacts (Associations)
- Tickets (Associations)
- Deals (Associations)
- Line items (Associations)

Voor het opslaan van deze “queueable” jobs, jobs die in een chain/bus achter elkaar gedraaid kunnen worden, worden 3 tabellen via migrations toegevoegd aan de database: “jobs”, “job_batches” en “failed_jobs”. Deze kunnen gegenereerd worden door Laravel, en is voor het project ook eenmalig gedaan. De Jobs tabel bevat een log van losse jobs die zijn uitgevoerd. Job_batches houdt alle batch jobs bij die zijn gedaan, dus ook de batch aan synchronisatie jobs, en failed_jobs slaat alle losse jobs op die zijn gefaald, met de reden waarom. In het geval van de synchronisatie wordt voornamelijk de job_batches tabel gevuld. Hier staat dan in hoeveel jobs er zijn uitgevoerd in een bepaalde batch, hoeveel er nog “pending” zijn en hoeveel er gefaald zijn.

4.4 Database

4.4.1 MySQL

Als database is gekozen voor MySQL. MySQL is een Open Source database systeem voor relationele databases. De reden dat voor MySQL is gekozen is allereerst omdat het erg goed samenwerkt met Laravel. Laravel ondersteund momenteel 4 databasesystemen: MySQL, Postgres, SQLite en SQL Server. Dit zijn allemaal relationele SQL databases, en dit is zo omdat Laravel onder water veel gebruik maakt van deze relaties in SQL, voornamelijk via de “Eloquent” modellen structuur. MySQL is in principe altijd de standaard keuze geweest als SQL database, en is in mijn geval ook gewoon een goede optie. SQLite was niet aan te raden omdat het slechter kan gaan werken bij een grotere/vollere database, aangezien het self-contained is en niet op een aparte server draait. SQL Server was ook geen ideale keuze omdat dit voornamelijk handiger werkt bij het gebruik van andere Microsoft omgevingen zoals Azure. Waar PostgreSQL een aantal voordelen biedt over MySQL, zoals het feit dat het een object-relatie database is en het meer complexere queries kan behandelen, biedt het geen functionaliteiten die nodig zijn voor de synchronisatietool wat MySQL niet kan. Daarom is voor de simpelste keuze gegaan die ook vanuit Laravel en CODE14 de voorkeur krijgt, namelijk MySQL.

4.5 Hosting

4.5.1 Ontwikkel

Voor de ontwikkeling van de synchronisatietool is het handig om het project lokaal te kunnen draaien, waarbij het eenvoudig en snel is om nieuwe code te kunnen testen. Het kost namelijk veel tijd en moeite om constant een project te deployen naar een server terwijl je aan het ontwikkelen bent, en dus niet zeker weet of de code gaat werken. Daarom wil je dus je wijzigingen live geupdate hebben, zodat je snel ziet wat je wijziging precies heeft gedaan en je dus sneller de juiste veranderingen kan aanpassen.

Voor het deployen van de code naar een lokale server zijn twee opties: hosten via een PHP development server op localhost, of hosten via Docker. De eerste optie is erg eenvoudig te realiseren door gebruik te maken van Laravel’s ingebouwde Artisan tools. Artisan commando’s worden voor veel Laravel functionaliteiten gebruikt, waaronder het aanmaken van specifieke classes (database model classes, controller classes etc.), het draaien van

migrations naar de database etc. Door het commando “php artisan serve” aan te roepen start Laravel een PHP development server op waarin het Laravel project gehost wordt. Wijzigingen die worden gedaan in het Laravel project worden dan ook live aangepast, dus het is niet nodig om de server opnieuw op te starten na elke wijziging. Daarbij kan met behulp van Webpack ook de front-end live worden gebuild na elke wijziging, zodat je ook de wijzigingen front-end live ziet updaten. Dit werkt omdat Webpack alle front-end bestanden bundelt naar een aantal vaste bestanden, en deze in een “public” folder plaatst. Alle CSS code wordt bijvoorbeeld samengevoegd in een enkel bestand, alle Javascript code wordt in een enkel bestand gezet etc. Die public map wordt dan gebruikt door Laravel om de webviews te serveren. Op deze manier gaat ontwikkelen en debuggen een heel stuk sneller dan als je constant je gehele project opnieuw moet draaien.

In plaats van het opzetten van een PHP development server via Artisan is er ook de optie om het project via Docker te hosten. Dit heeft als voornamelijkste voordeel dat je geen lokale MySQL server moet opzetten via bijvoorbeeld XAMPP: dit kan rechtstreeks in een Docker container worden gedeployed. Daarbij heeft CODE14 een speciaal Docker project opgezet die ervoor kan zorgen dat je je project kan routen naar een specifieke URL, via Traefik. In plaats van dat je project dan lokaal draait op een localhost url kan deze draaien op een eigen url, zoals hubspot-synctool.dev.test. Vervolgens kan je op dezelfde manier doorontwikkelen als met een lokale PHP server. Door hier ook Webpack te laten draaien kan je op dezelfde manier ook de front-end updates live zien veranderen. Voor het bouwen van de tool is het draaien van een lokale PHP server iets handiger, voornamelijk omdat deze iets sneller is. Het is echter wel goed om af en toe de docker versie te checken, omdat het kan zijn dat bepaalde functionaliteiten in de ene versie wel of niet werken.

4.5.2 Test

De testomgeving van de synchronisatietool wordt gehost via Laravel Forge. Laravel Forge is een server management en web-deployment service, ontwikkeld door en voor Laravel. Laravel Forge host zelf de servers, die aan te schaffen zijn d.m.v. een abonnement. CODE14 heeft hier een abonnement op, en kan daarmee oneindig veel servers/domeinen aanvragen. De webserver die gebruikt wordt is Nginx. De reden dat ik dit ook gebruik is omdat dit zo’n beetje de standaard is voor testomgevingen binnen CODE14. Een andere hosting service was uiteraard mogelijk, maar met Laravel Forge is het meeste ervaring en is uiteraard ook gespecialiseerd op het hosten van Laravel projecten. Het project zelf is echter nog niet zo ver gekomen dat het nodig was om een deployment te doen naar een testserver. Dit is namelijk eigenlijk pas nodig op het moment dat vrijwel alle synchronisaties zijn geïmplementeerd en de tool echt goed getest kan worden door verschillende ontwikkelaars. Dit betekend dus ook dat de hosting op acceptatie en productie ook nog niet van toepassing zijn geweest.

4.5.3 Acceptatie & Productie

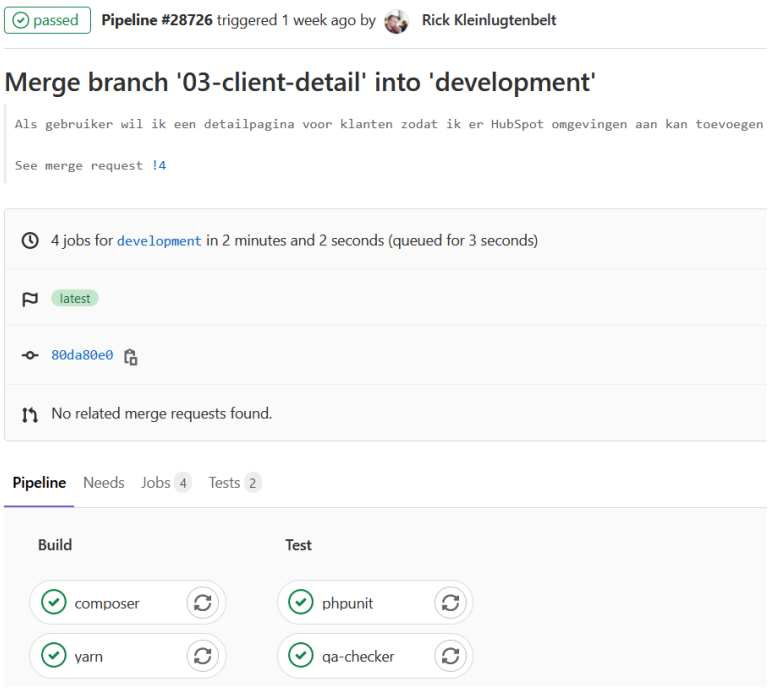
Voor de hosting van de acceptatie- en productieversie van de synchronisatietool wordt gebruik gemaakt van de servers van Shockmedia. Shockmedia is een Nederlandse provider van hosting services, en wordt door CODE14 gebruikt om vrijwel al haar projecten (acceptatie en productie) in te hosten. De synchronisatietool zal hier op een Shared VPS

komen te draaien, samen met een aantal andere kleinere projecten. Bij grotere projecten met meer verkeer is het namelijk belangrijk dat deze op een dedicated server komt te staan, maar bij een project als de synchronisatietool is dit overbodig en zou veel extra kosten met zich meebrengen. Ook Shockmedia is vrijwel altijd de standaard bij acceptatie en productieomgevingen van projecten van CODE14. Ook hier was een andere service mogelijk voor mijn project, maar om voor een veiligere keuze te gaan waar meer ervaring mee is binnen CODE14 heb ik ook gekozen voor Shockmedia.

4.6 CI/CD

Elke keer als er een branch wordt gepusht naar Gitlab, of als er een merge request wordt aangemaakt/gemerged draait er een pipeline. Deze pipeline bestaat uit 2 stages: een “build” stage en een “test” stage. De build stage bevat twee build jobs: een “composer” build en een “yarn” build. De composer job draait onder andere composer install, die gaat verifiëren of alle PHP packages correct geïnstalleerd kunnen worden. De yarn build doet hetzelfde voor de NPM packages in package.json. Beide jobs en de stage zijn succesvol als er geen errors ontstaan bij het installeren.

De test stage bestaat ook weer uit 2 jobs: een “phpunit” job en een “qa-checker” job. De phpunit job start een lokale database op en draait een migration. Vervolgens gaat hij op zoek naar Unit tests die zijn gedefinieerd in het project, en deze gaat hij vervolgens een voor een uitvoeren. Als de database migrations goed werken en de Unit tests allemaal een goed resultaat teruggeven slaagt de job. De laatste job is de qa-checker job. Deze job maakt gebruik van de “qa” package, gemaakt door ontwikkelaars bij CODE14 zelf. Deze package kan het project doorspitten om het project te checken op problemen, netheid foutjes etc. Meer daarover onder het kopje QA. Deze job slaagt als de checker geen problemen herkent in de gewijzigde code. Als alle jobs/stages zijn geslaagd, dan geeft is de pipeline succesvol geslaagd. Als echter al 1 job faalt dan is de hele pipeline gefaald.



Figuur 18: Screenshot van een pipeline in Gitlab

4.7 Testen

4.7.1 Unit Tests

Het testen van de functionaliteiten van de applicatie wordt voornamelijk gedaan door het uitvoeren van PHP Unit Tests. PHP unit tests bestaan uit relatief simpele functies die een enkel doel hebben, zoals het testen van een API. Je stuurt dan een request naar een specifieke URL, en vervolgens “assert” je of de response status goed is en/of je het juiste object terug krijgt.

Deze tests worden ook uitgevoerd in de build pipeline op Gitlab. Deze vereist dat alle tests correct slagen, anders faalt de job en dus ook de pipeline. De unit tests die zijn geschreven voor de synchronisatietool roepen verschillende API’s aan van de Clients en Accounts, en wordt een API van HubSpot aangeroepen. Ook wordt er een test API uitgevoerd die verwacht een fout resultaat te krijgen, namelijk het ophalen van een specifieke client die niet bestaat.

4.8 QA

4.8.1 Branching strategie

Om de kwaliteit van de code goed te kunnen bewaren wordt er voor elke issue/user story die wordt geïmplementeerd een nieuwe branch aangemaakt. Zodra de story is geïmplementeerd wordt de functionaliteit en de kwaliteit van de code gecontroleerd door een andere ontwikkelaar uit het team. Dit wordt opgezet door een merge request aan te maken voor de feature branch terug naar de development branch. Bij deze merge request kan dan aan een andere ontwikkelaar worden gevraagd om de merge request te reviewen. Deze kan dan de gegeven branch uitchecken, en via de source control (GitLab) precies zien wat er veranderd is, en daar dan feedback op geven. Dit resulteert dus in een goedkeuring van de andere ontwikkelaar, of een afwijzing met feedback. Deze moet dan eerst verwerkt worden voordat hij ge-merged kan worden.

4.8.2 QA Checker

Naast de vereiste manier van werken voor de source control (branching, merge requests etc.) is er door CODE14 ook een speciale QA tool ontwikkeld, genaamd de QA-checker. De QA-checker is voornamelijk een verzameling van allerlei code controle software, die een project kunnen doorspitten op zoek naar bijvoorbeeld beveiligings-issues, verkeerde syntax, overbodige console-logs of uitgecommende code enzovoorts. Een voorbeeld van zo’n package is PHP CodeSniffer: deze controleert PHP code op bepaalde code-standaarden zoals naamgeving/casing of PHPDoc. Voor Javascript is er bijvoorbeeld ESLint: dit is een veelgebruikte “linter”, die ook voornamelijk kwaliteit en codeer stijl problemen herkent zoals het gebruikt van enkele/dubbele quotes, het toevoegen van een lege lijn aan het eind van een bestand enzovoorts. Daarbij is er door CODE14 ook een specifiek code-style bestand

gemaakt voor allerlei bestandstypes om bijvoorbeeld regels op een bepaalde manier te uitlijnen (bepaald aantal spaties per indent), het gebruiken van enkele vs. dubbele quotes enzovoorts. Daardoor zien alle projecten er qua layout er precies hetzelfde uit.

De QA-checker kan op meerdere manieren worden gebruikt. Het project kan lokaal worden geïnstalleerd om daar je project rechtstreeks te controleren, of het kan worden ingebouwd in de pipeline van een repository. Deze draait dan elke keer als er code wordt gepushed naar een branch en geeft vervolgens direct de resultaten weer. Als er ergens iets niet klopt zorgt de QA-checker ervoor dat de pipeline faalt, en deze geeft dan duidelijk weer waar in de code er problemen zitten. Daarbij geeft het ook vaak aanbevelingen hoe dit opgelost kan worden, en in sommige gevallen kan dit zelfs automatisch door de QA-checker worden gedaan. Hieronder is te zien hoe een resultaat van zo'n QA-check er ongeveer uit kan zien:

```
$ qacheck-ci
Checking changes against target branch origin/development...
┌ /app/Http/Kernel.php
|
|   35  This comment is 55% valid code; is this commented out code?
|       (phpcodesniffer: Squiz.PHP.CommentedOutCode.Found)
|
└ /app/Http/Kernel.php
┌ /app/HubSpot/HttpClient.php
|
|    9  Class constants must be uppercase; expected BASEOAUTHURI but found
|       baseOAuthUri
|       (phpcodesniffer:
|       Generic.NamingConventions.UpperCaseConstantName.ClassConstantNotUpperCas
|       e)
|
|   35  Method \App\HubSpot\HttpClient::getAccessToken() does not have parameter
|       type hint nor @param annotation for its parameter $redirectUri.
|       (phpcodesniffer:
|       SlevomatCodingStandard.TypeHints.TypeHintDeclaration.MissingParameterTyp
|       eHint)
|
```

Figuur 19: Screenshot van QA resultaten uit pipeline

In dit geval is er dus een uitgecommende regel gevonden, is de naamgeving van een bepaalde variabele niet juist en mist er ergens een PHPDoc annotatie voor een parameter in een functie. De getallen die ervoor staan (35, 9, 35) wijzen precies naar het regelnummer in het bestand waar het probleem is gevonden, zodat je eenvoudig het probleem kan opzoeken en oplossen.

5 Procesbeschrijving

5.1 Plan van Aanpak

De eerste weken van de afstudeerstage stonden in het teken van het schrijven van een Plan van Aanpak en het beter leren kennen van het bedrijf. Deze weken stonden ook in het teken van het beter begrijpen van de afstudeeropdracht, het leren kennen van HubSpot en van de technische stack die bij CODE14 wordt gebruikt (Laravel & Vue.js). Uit het schrijven van dit Plan van Aanpak zijn een aantal resultaten gekomen. Allereerst zijn de onderzoeksvragen voor de onderzoeksfase opgesteld, zodat daar voorbereid aan begonnen kon worden. Verder is er gekeken naar hoe CODE14 zelf als bedrijf functioneert en is er beschreven hoe de verschillende fases van het afstuderen er ongeveer uit gaan zien (opstartfase, onderzoeksfase, ontwikkelfase en afrondfase). Ten slotte zijn de verschillende werkmiddelen en technieken die (waarschijnlijk) gebruikt zouden worden beschreven. Dit alles was handig om te doen omdat er dan met een betere blik begonnen kon worden met het uitvoeren van de daadwerkelijke afstudeeropdracht.

5.2 Onderzoek/ontwerp

Na het schrijven van een Plan van Aanpak voor de rest van de afstudeerperiode was het tijd om een onderzoek te doen. Om goed voorbereid aan de slag te kunnen met het bouwen van de applicatie moesten er een aantal onderzoeksvragen beantwoord worden. De hoofdvraag voor het onderzoek was: “Hoe krijg ik een generieke, configureerbare synchronisatietool waarbij data van een klant uit de productieomgeving in HubSpot geanonimiseerd gesynchroniseerd wordt over de OTAP?”. Dit is een vrij lange onderzoeksvraag, maar benoemd wel alle belangrijke punten waar de synchronisatietool aan moet voldoen. De tool moet generiek zijn in de zin dat de tool zonder wijzigingen in de code voor elk HubSpot account te gebruiken moest zijn. Dit omdat je niet wil dat je bij elk nieuwe account waar je een synchronisatie mee wilt doen je eerst via de code de instellingen van het account moet gaan invullen. Dit duurt allereerst erg lang voordat de nieuwe code weer terug is bij productie, en het toevoegen van nieuwe accounts gebeurd nog wel regelmatig. Vandaar moest de synchronisatietool dus zo veel mogelijk configurabel zijn vanuit de front-end. Het tweede deel van de hoofdvraag gaat over de synchronisatie zelf, namelijk dat je data uit de HubSpot productieomgeving van een klant wilt synchroniseren naar 1 of meerdere HubSpot testaccounts van ontwikkelaars bij CODE14. Omdat dit over data gaat van onder andere echte personen/bedrijven is het nodig om een deel van deze data te anonimiseren tijdens de synchronisatie. “over de OTAP” gaat dus niet over de OTAP van de synchronisatietool zelf, maar over het feit dat de synchronisatietool data moet kunnen synchroniseren tussen de productieomgeving van een klant in HubSpot naar een ontwikkel/test/acceptatie omgeving van een testaccount in HubSpot van een ontwikkelaar bij CODE14.

De hoofdvraag is vervolgens opgedeeld in 7 deelvragen, die elk een onderdeel van de synchronisatie behandelen. Het onderzoek is voornamelijk uitgevoerd door verschillende documentatie door te lezen, denk aan de HubSpot documentatie (CRM API | Understanding the CRM., z.d.), verschillende anonimiseringstechnieken (Anonimiseren en pseudonimiseren., z.d.) en technische documentaties van bijvoorbeeld Laravel (Encryption,

z.d.). Dit was nodig om een goed beeld te krijgen van wat er precies mogelijk was voor de synchronisatie: hoe werkt het ophalen en invullen van data in HubSpot, hoe kan ik het beste de data anonimiseren, hoe encrypt ik mijn tokens als ik deze op wil slaan in de database etc. Met name de HubSpot documentatie was daar belangrijk, omdat hier duidelijk werd wat de synchronisatietool allemaal wel en niet kon synchroniseren, en wat de limieten zijn aan het aanroepen van de HubSpot API's. Verder is er ook veldwerk gedaan binnen HubSpot, om goed te kunnen begrijpen wat je nou precies met HubSpot kan. Hier is gekeken naar alle losse pagina's binnen HubSpot en de functionaliteit daarvan, maar ook naar de aanpasbaarheid van de verschillende data componenten. Ten slotte is er gesproken met meerdere ontwikkelaars binnen CODE14 die ervaring hebben met HubSpot, en dan met name het bouwen van de koppelingen tussen een klantsysteem en HubSpot. Dit was handig omdat zij vaker hebben gewerkt met de API's van HubSpot, en omdat zij ook goed konden vertellen welke functionaliteiten veel gebruikt werden binnen HubSpot en welke minder.

Naast het onderzoek was het handig om alvast bepaalde zaken visueel en technisch uit te werken. Om een goed idee te krijgen van hoe de front-end er ongeveer uit moest gaan zien en welke data er beschikbaar moest komen zijn wireframes gemaakt. Daarnaast is er een database diagram uitgewerkt om een goed idee te hebben van welke tabellen er nodig zullen zijn en hoe deze aan elkaar gekoppeld zullen worden. Verder is er een architectuurplaat uitgewerkt om de verschillende acties/functionaliteiten van de synchronisatietool weer te geven, om aan te geven hoe de verschillende OTAP omgevingen gehost worden en hoe de koppelingen zitten tussen verschillende technische componenten als de database, HubSpot en de synchronisatietool zelf. Ten slotte is er nog een soort flowchart uitgetekend om aan te geven hoe de flow van de synchronisatie zelf gaat. Dit was allereerst handig omdat de volgorde waarop verschillende objecten uit HubSpot gesynchroniseerd moeten worden belangrijk is, aangezien sommige objecten afhankelijk zijn van andere. Daarnaast was het handig om uit te denken hoe een individuele synchronisatie taak ongeveer moest lopen, zodat dat over de verschillende objecten ongeveer op dezelfde manier geïmplementeerd kan worden.

5.3 Ontwikkeling

5.3.1 SCRUM

Tijdens het ontwikkelen is tot in zekere mate gewerkt volgens de SCRUM methode. Het team waar ik in heb gewerkt draait sprints van 1 week. De sprints begonnen op maandag en eindigen op vrijdag. Elke vrijdag is er ongeveer anderhalf uur uitgetrokken om nieuwe User Stories te schatten. Dit is een ruim tijdsvak omdat de meeste ontwikkelaars aan verschillende projecten werken, en ze dus soms voor meer projecten stories moeten schatten. Na de schatting is er een plannings meeting waar iedereen hun uren doorgeeft en aangeeft met welke projecten/stories ze bezig gaan. Ten slotte is er om de week een retrospective waar de dingen worden besproken die in de weken daarvoor goed en slecht zijn gegaan, en wat ze daar aan kunnen veranderen, wat ze moeten stoppen en wat ze moeten blijven doen.

Omdat ik zelf aan mijn eigen project werk en er geen "budget" hangt aan mijn uren/planning ben ik in de meeste gevallen wel aanwezig bij de meetings, maar werk ik niet met een vaste

weekplanning m.b.t. mijn stories. Daarbij schat ik ook zelf het aantal story points per story in. Ik hou daarbij wel een Scrumboard bij waar ik mijn User stories inzet en bij hou hoe ver elke story is (To-Do, Doing, For Review, Done). Ook werkt het team niet met een klassieke burn-down per sprint, maar eerder op basis van een budget. Projecten voor een klant zijn meestal een langer doorlopend project, en het verschilt vaak per week wie er met het project bezig gaat en hoeveel er gedaan moet worden. Sommige ontwikkelaars zijn bijvoorbeeld langer bezig met een story van 3 punten dan andere, en deze hebben dus een ander “budget”. Vandaar dat storypoints ook niet globaal zijn gelinkt aan een aantal uren, maar dit verschilt per ontwikkelaar.

5.3.2 User stories

De user stories voor het bouwen van de synchronisatietool zijn in eerste instantie opgedeeld in 3 Epics:

- HubSpot klant configuraties
- HubSpot datasynchronisatie
- Synchronisatietool overig

De reden voor deze opdeling is omdat de configuratie van de klanten en de HubSpot accounts die onder deze klanten horen en de synchronisatie tussen twee HubSpot accounts binnen de synchronisatietool weinig met elkaar te maken hebben. Het zijn aparte pagina’s waar beide functionaliteiten gebeuren, en beide kunnen ook los van elkaar gebeuren. Daarnaast zijn er nog een aantal overige stories die te klein zijn of waar er te weinig bij elkaar horen om er een eigen epic van te maken. Dit gaat bijvoorbeeld over het schrijven van tests, het afschermen van de applicatie met bijv. een login etc. Een kleinere opdeling was eventueel mogelijk, door bijvoorbeeld front- en back-end taken op te delen, maar dan werden beide epics ook weer vrij klein, dus dit was ook overbodig.

Klant configuraties duidt op alles wat een gebruiker moet kunnen doen in de UI van de synchronisatietool om klanten met de bijbehorende HubSpot accounts te kunnen registreren. Hieronder vallen dus stories als het opzetten van een overzichtspagina voor klanten, een detailpagina voor accounts van klanten en het verwerken van de nodige gegevens.

HubSpot data synchronisatie gaat over alles wat te maken heeft met de daadwerkelijke synchronisatie. De stories onder deze epic zijn voornamelijk opgezet op basis van de verschillende data-objecten die moeten worden gesynchroniseerd. De synchronisatie van contacten en de synchronisatie van “companies” zijn bijvoorbeeld aparte user stories. Ook moet er onder deze epic een pagina worden gebouwd waarin de synchronisatie gestart kan worden, en bij voorkeur ook gevolgd kan worden (hoe ver is de synchronisatie, hoe lang duurt het nog, waar zijn fouten opgetreden etc.).

“Synchronisatietool overig” bevat alle stories die niet persé onder een grote epic vallen. Op dit moment is dat bijvoorbeeld het afschermen van de synchronisatietool met een inlogsysteem.

Elke user story krijgt een aantal story points die aangeven hoeveel werk het ongeveer gaat kosten om de story te implementeren. Hierbij heb ik ervoor gekozen om aan het begin van

de ontwikkeling maar een paar stories een gewicht te geven. Dit omdat ik nog niet zeker wist hoe lang alles precies zou duren, en omdat ik verwachtte dat de eerste paar stories langer gingen duren i.v.m. het feit dat ik nog niet veel ervaring had met Laravel & Vue.js. Hierdoor zou er bijvoorbeeld meer feedback te verwerken zijn uit Merge Requests. Zodra het tijd was om bezig te gaan met nieuwe stories heb ik weer de volgende aantal stories punten gegeven enzovoorts. Daarbij heb ik een schema bijgehouden met alle stories, de dagen waarop ik er aan was begonnen, de dag waarop ze klaar waren, het aantal punten dat de story had gekregen en hoeveel uur ik er ongeveer mee bezig was. Dit schema gaf mij een redelijke indicatie van hoeveel punten een story relatief gezien ongeveer zou moeten krijgen, maar liet ook zien wat de verhouding ongeveer was tussen sommige schattingen. Dit schema is te vinden in de bijlagen.

6 Conclusie

6.1 Onderzoek

De hoofdvraag van het onderzoek was “Hoe krijg ik een generieke, configureerbare synchronisatietool waarbij data van een klant uit de productieomgeving in HubSpot geanonimiseerd gesynchroniseerd wordt over de OTAP?”. Om deze vraag te beantwoorden zijn meerdere deelvragen opgesteld, en er zijn dus ook meerdere antwoorden om deze hoofdvraag te beantwoorden.

De tool kan generiek worden ingezet voor verschillende klanten en omgevingen door de tool in te zetten op een centrale server, en het mogelijk te maken om klanten en HubSpot accounts te kunnen aanmaken. Vervolgens kan voor elk account de synchronisatie op dezelfde manier worden gestart zonder dat extra configuraties nodig zijn, omdat alle variabele data uit HubSpot via API’s kan worden opgehaald en kunnen worden verwerkt. Zo maakt het dus van tevoren niet uit welke velden er bijvoorbeeld actief zijn bij een bepaald dataobject: dit kan allemaal uit de API’s van HubSpot worden achterhaald. Ook de verschillende pakketten en rate limits zouden geen probleem moeten zijn om een synchronisatie uit te voeren, bijvoorbeeld omdat fouten van de limieten op dezelfde manier worden behandeld en omdat de objecten die gesynchroniseerd moeten worden in HubSpot hetzelfde zijn in de verschillende pakketten.

Anonimisering gebeurd door het inzetten van een data-faker. In eerste instantie is er geprobeerd om de privacy eisen gelijk te stellen over alle klanten heen. Zoveel mogelijk velden/properties die van tevoren bekend zijn worden geanonimiseerd. In een nieuwere versie zouden ook de niet-bekende velden/properties, zoals properties die worden aangemaakt door klanten zelf, ook geanonimiseerd kunnen worden door deze apart te configureren binnen de tool.

Ten slotte moest de tool over de hele OTAP van een ontwikkelaar bij CODE14 beschikbaar komen. O/T en A zijn voor de tool in dit geval echter vrijwel gelijk. Voor deze omgevingen worden in HubSpot test-accounts aangemaakt, en deze zijn allemaal identiek aan elkaar. Ze verschillen dus eigenlijk alleen van naam, en van doeleinde voor de ontwikkelaar. In de tool kan hier rekening mee gehouden worden door de productie accounts van een klant in HubSpot apart te registreren in de tool van de overige accounts. Vervolgens kan dat productie account bij het starten van de synchronisatie niet geselecteerd worden als doel-account.

6.2 Implementatie

Voor de implementatie van de synchronisatietool is een webapplicatie gerealiseerd m.b.v. het Laravel PHP framework als backend en Vue.js als frontend. In deze applicatie kunnen klanten worden geregistreerd via een UI, en onder deze klanten kunnen vervolgens HubSpot accounts/omgevingen worden geregistreerd. Onder deze accounts zit een scheiding tussen een productie-account en overige accounts. Het verschil daartussen is dat een productie-account niet als doel-account kan worden gebruikt bij de synchronisatie, omdat je

alleen maar wil synchroniseren vanuit een productie account naar een ontwikkel account. Bij het aanmaken van een account heeft de gebruiker een optie om een API key te registreren, of aan te geven dat het account gebruik maakt van OAuth. In dat geval wordt de gebruiker doorverwezen naar HubSpot om toestemming te geven tot het gebruik van OAuth tokens door de applicatie, en vervolgens kan de tool voor dat account access/refresh tokens ophalen. Al deze gegevens worden opgeslagen in een MySQL database, waarbij de API key en de OAuth tokens worden ge-encrypt.

Daarnaast is er een pagina gerealiseerd om een synchronisatie te starten. De synchronisatie bestaat op het moment alleen nog uit het synchroniseren van properties met de bijbehorende property groups (property = bijv. email, property group = bijv. contact_information). Daarbij kunnen er ook contact objecten worden gesynchroniseerd, waarbij de nodige variabelen worden geanonimiseerd door te kijken naar welke properties worden gebruikt in dat object. Het is echter niet veel extra werk om de overige CRM objecten te implementeren, omdat deze dezelfde principes volgen als de contact-objecten. De responses uit de API's zijn dan ook altijd op dezelfde manier opgebouwd, dus dat is alleen een kwestie van API's aanpassen en nieuwe transformers/fakers implementeren. Het is echter wel zaak dat alles goed getest en nagekeken wordt, en daarom duurt het vaak langer om de individuele stories/functionaliteiten echt af te ronden. Vandaar ook dat er nog niet veel is afgerond aan de synchronisatie zelf.

Ten slotte zijn er nog een aantal zaken die in de toekomst ook wenselijk zijn om in te bouwen in de synchronisatietool. Als het project uiteindelijk in productie wordt gebruikt zal het moeten worden afgeschermd met een inlogsysteem. Daarbij is het dan ook nodig om te zorgen dat de rate-limits niet tot zijn max worden geraakt, omdat dit het aanroepen van API's vanuit alle kanten belemmerd. Ten slotte zou het mooi zijn om in de tool extra feedback te geven op hoe ver de synchronisatie is, en of er dingen zijn die bijvoorbeeld fout zijn gegaan.

7 Aanbevelingen

7.1 Configureerbare velden

Een van de functionaliteiten die voornamelijk nog mist in de synchronisatietool is de mogelijkheid om individuele datavelden te registreren voor o.a. anonimisering. Zoals eerder is beschreven zijn de CRM objecten binnen HubSpot (contact objecten, deal objecten etc.) op verschillende manieren aanpasbaar. Dit gaat voornamelijk over de properties: de datavelden die gekoppeld zitten aan een CRM-object. Zo'n property is bijvoorbeeld een e-mail veld bij een contact. Van die properties zijn er een hele hoop, maar standaard staan er maar een aantal aan bij elk object. Die properties kunnen dan los worden aangezet of uitgezet, en er kunnen dus ook eigen properties worden aangemaakt. De synchronisatietool kan deze lijsten van properties wel synchroniseren van account naar account, maar het probleem ontstaat bij het ophalen van de inhoud van deze properties bij elk CRM-object. Hiervoor moet je namelijk in de API meesturen welke property-data je van elk object wilt hebben. Zoals ook is onderzocht onder het onderwerp "variabele datastructuur" zijn dit er bij elk object heel veel, en moet de synchronisatietool er dus zelf achter komen welke properties daadwerkelijk aan staan om ze op te vragen. Dit kan grotendeels door te kijken naar de verschillende variabelen die worden bijgehouden in een property-object, zoals of de property "hubspotDefined" is (gedefinieerd door HubSpot), of dat het object "calculated" is (een waarde die automatisch berekend wordt door HubSpot).

Dit is geen ideale situatie. Het probleem hier zit hem namelijk vooral in de anonimisering van deze properties. Het project zelf weet nu maar van een aantal properties wat voor soort veld het is, en dus wat voor soort data hij moet genereren om de inhoud ervan te vervangen. Een naam moet een andere naam worden, een straatnaam een andere straatnaam etc. Als een klant zelf een eigen property heeft aangemaakt, of een property heeft aangezet die het project niet kent kan deze dus ook nog niet geanonimiseerd worden.

Een oplossing voor dit probleem is het implementeren van een soort veld-configuratie systeem, dan kan worden toegepast onder elk geregistreerde HubSpot account in de synchronisatietool. Het idee hier is dat een ontwikkelaar eenmalig de velden kan registreren onder de te synchroniseren HubSpot objecten, waarbij aan wordt gegeven wat voor soort dataveld dit is. Op die manier zou de synchronisatietool precies weten welke velden hij moet opvragen bij elk object en hoe hij die moet anonimiseren (of juist niet).

8 Inhoudelijke reflectie

Het onderzoek dat is uitgevoerd voor deze afstudeeropdracht is voornamelijk gefocust op hoe de synchronisatietool uiteindelijk gebouwd moet gaan worden, waar de complexiteiten daar in zitten en hoe bepaalde problemen opgelost moeten worden. Bepaalde zaken die daarbij zijn onderzocht bleken later echter relatief overbodig. Er is bijvoorbeeld vrij uitgebreid gekeken naar de individuele API's voor elk CRM-object, wat eigenlijk niet erg relevant meer was voor het bouwen van de applicatie omdat wel duidelijk werd dat de meeste API's voor de synchronisatie wel beschikbaar waren. Het was hier beter geweest om bijvoorbeeld meer te focussen op een aantal specifieke API's zoals die voor het ophalen van de properties, omdat dit de meest complexe waren en de response objecten het grootste, en ook omdat deze toch heel belangrijk zijn voor het uitvoeren van de rest van de synchronisatie. Er had dus eerder aan het bouwen van de applicatie begonnen kunnen worden als het onderzoek iets beknopter was geweest.

Daarnaast is het maken van de wireframes en de database diagram wel erg behulpzaam gebleken. Dit geldt ook voor de flowchart waarin de volgorde en functionaliteiten van de synchronisatie duidelijk werden. Deze zijn dan ook goed gebruikt tijdens het bouwen van de synchronisatietool. Ook hebben alle onderzoeksvragen zeker nieuwe vragen maar ook meer duidelijkheid naar boven gebracht. Het onderzoek naar de functionele werking van HubSpot was cruciaal voor het begrijpen van het nut van de tool, en een oplossing voor het verwerken van de variabele datastructuur was ook nodig om duidelijk te kunnen krijgen of het bouwen van zo'n synchronisatietool wel echt de juiste oplossing was voor dit probleem. Voor de gebruiksvriendelijkheid van de synchronisatietool en het zo veel mogelijk helpen van ontwikkelaars bij het uitvoeren van zo'n synchronisatie is dit wel de beste oplossing omdat dit de minste handmatige acties vereist. Als het echter extra belangrijk wordt dat van tevoren precies duidelijk moet zijn welke velden gesynchroniseerd moeten worden en welke niet, en op welke manier deze geanonimiseerd moeten worden, dan had de tool er waarschijnlijk anders uit gezien en had de focus misschien meer gelegen op het genereren van data voor een omgeving op basis van de aangemaakte properties/velden. Uiteindelijk zou een combinatie tussen beide het beste zijn, maar de vraag is dan dus hoeveel werk je een ontwikkelaar alsnog wil laten doen voor het invullen van al die velden.

Het bouwen van de applicatie zelf kostte uiteindelijk meer tijd dan gepland. Helemaal omdat er vooraf nog weinig ervaring was met Laravel en Vue kostte het implementeren van de eerste paar stories veel tijd. Ook gingen er vaak meerdere dagen overheen voordat een story echt af was i.v.m. het nakijken van merge requests. Als deze bijvoorbeeld aan het eind van de dag werden nagekeken, kon deze dus pas de volgende dag worden verbeterd en gaan er al snel meerdere dagen overheen voordat het echt klaar is. Daarbij waren veel stories al snel afhankelijk van bepaalde functionaliteiten uit andere stories, dus dit was lastig om om heen te werken. Dit ging echter uiteindelijk wel een stuk soepeler na een aantal stories.

9 Versiebeheer

Versie	Datum	Omschrijving
1.0	26-03-2021	1e versie
1.1	10-05-2021	Versie 1.1, aangeleverd voor tussentijdse meeting
2.0	30-05-2021	Concept versie
3.0	13-06-2021	Definitieve versie

10 Literatuurlijst

Anonimiseren en pseudonimiseren. (z.d.). Considerati. Geraadpleegd op 17 maart 2021, van https://www.considerati.com/nl/kennisbank/anonimiseren-en-pseudonimiseren.html

CRM API | Understanding the CRM. (z.d.). HubSpot. Geraadpleegd op 9 maart 2021, van https://developers.hubspot.com/docs/api/crm/understanding-the-crm

Documentation - Tailwind CSS. (z.d.). Tailwindcss. Geraadpleegd op 14 april 2021, van https://tailwindcss.com/docs

Eloquent. (z.d.). Laravel. Geraadpleegd op 22 april 2021, van https://laravel.com/docs/8.x/eloquent

Encryption. (z.d.). Laravel. Geraadpleegd op 18 mei 2021, van https://laravel.com/docs/8.x/encryption

FakerPHP / Faker. (2021, 30 maart). GitHub. Geraadpleegd op 21 mei 2021, van https://fakerphp.github.io/

HubSpot APIs | Client Libraries. (z.d.). HubSpot. Geraadpleegd op 19 mei 2021, van https://developers.hubspot.com/docs/api/client-libraries

HubSpot APIs | Usage guidelines. (z.d.). HubSpot. Geraadpleegd op 9 maart 2021, van https://developers.hubspot.com/docs/api/usage-details

Introduction to Vue.js. (z.d.). Vue.js. Geraadpleegd op 14 april 2021, van https://vuejs.org/v2/guide/

Webpack - Concepts. (z.d.). Webpack. Geraadpleegd op 14 april 2021, van https://webpack.js.org/concepts/

Working with OAuth | OAuth Quickstart Guide. (z.d.). HubSpot. Geraadpleegd op 19 maart 2021, van https://developers.hubspot.com/docs/api/oauth-quickstart-guide

Zaninotto, F. (2020, 27 oktober). Faker. GitHub. https://github.com/fzaninotto/Faker

11 Bijlagen

Bijlage 1: V2.0 Onderzoeksrapport (extern document)

Bijlage 2: User Stories (extern document)

Bijlage 3: Zelfreflectie (extern document)